COMMODORE MAGAZINE

VOL 2
ISSUE 4

★ 6845 CRTC CONTROLLER

★ CBM COMPILERS

REGISTERED FOR POSTING AS A PUBLICATION: CATEGORY B

The objective of this magazine is to disseminate information to all users of Commodore computer products. This magazine contains a variety of information collected from other Commodore publications, and generated locally. Contributions from all Commodore User Groups, and individual users are encouraged.

Advertising space is restricted and allocated on a first-come, first-served basis. Advertising rates can be supplied on request.

All copy and advertising should be addressed to:

The Editor,
COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON
N.S.W. 2064
AUSTRALIA

ISSUE No.	COPY/ADV DEADLINE	PUBLICATION DATE
1	February 18th	March 6th
2	April 1st	April 17th
3	May 13th	May 29th
4	June 17th	July 3rd
5	July 29th	August 14th
6	September 9th	September 25th
7	October 14th	October 30th
8	November 25th	December 4th

Production & Typesetting: Mervyn Beamish Graphics Pty. Ltd.
82 Alexander Street, Crows Nest
2065
Phone 439 1827

Printer: Liberty Print
108 Chandos Street, Crows Nest
2067
Phone 43 4398

SUBSCRIPTIONS

	Annual Subscription	Single Copy
Postage paid within Australia	$A30.00	$A5.00
Overseas Postage Paid	$A38.00	$A6.00

Subscriptions to COMMODORE MAGAZINE can be obtained from your authorised Commodore Dealer, or from:

COMMODORE MAGAZINE,
P.O. BOX 336, ARTARMON,
N.S.W. 2064,
AUSTRALIA.

Vol 1 1981
Vol 2 1982

Typeset and assembled off Commodore Wordcraft disks

PLEASE NOTE: To provide a good information service to Commodore users, we will regularly mention equipment, software and services offered by companies and individuals not directly related to Commodore. In doing so, we are not making recommendations, and cannot be responsible for the validity and accuracy of any statements made.

EDITOR'S DESK . . .

The middle of the year races by and unfortunately the publication date of Issue 4 suffered. We appologise for this delay and are trying to get all the future publications back on schedule.

Commodore has been hectic with activity in the month of August, due to the participation at a number of shows throughout the country.

The biggest for Commodore was at the Data 82 show held at Centerpoint in Sydney, where Commodore with 10 stands full of product, made a big impression on the many thousands of visitors who came to see the many computers available.

The National Computer Conference in Hobart at the end of August was certainly the highlight of the 'computer year' with many overseas vistors attending. The Commodore presence was again highlighted with a stand showing the vast range of product and the flexibility in configuration.

There are many brands of computer now available on the market, but the suppliers such as Commodore still dominate with a vast range of both hardware and software than can be configured to suit most applications and budget.

If you would like further information on what is available from Commodore, read the Commodore News on page 2 for details on how to obtain the newspaper published by Commodore.

table of contents

VOLUME 2 ISSUE 4

Page	Contents
2	Commodore News
4	Letters to the Editor
5	Review-VIC 20 Programmers Reference Guide
6	Computers join the Work Force
7	Everything you wanted to know about the Commodore range of products - and asked!
12	A little VIC Music
14	6845 CRTC Controller
15	The Kernal
17	Compilers-What/Where
19	VIC Loader for CBM & PET Computers
20	VIC Magician
24	Window Text
26	BASIC Plotter
27	The SUPERPET
30	Using the Commodore Printer
32	VIC Operating System Maps
36	VIC Kernal Entry Points
25	Waterloo Micro Software

next issue:

VIC Memory Block Maps
Commodore's Range of Disk Drives
VIC Programming Aids
Fast Forward Load/SAve on VIC Datasette

list of Advertisers

B.S Microcomp	inside Cover
Compute CBM Systems	Back Cover
C.W. Electronics	Page 31
Mervyn Beamish Graphics	Page 31
Micropro Design	Page 31
Pittwater Computer Sales	Page 18
The Microcomputer House	Page 16

COMMODORE NEWS

VIC 20
An all-purpose reference guide for first-time computerists as well as experienced programmers!
PROGRAMMERS REFERENCE GUIDE
commodore COMPUTER
VM110

VIC 20 PROGRAMMERS REFERENCE GUIDE AVAILABLE

This valuable publication is now available from your Commodore Dealer. The recommended retail price is $22.00.

A review of this publication is on page 5 of this issue.

PET CLEANING KITS NOW AVAILABLE

A new product recently announced, is the PETKIT cleaning system for the Commodore range of computers.

The kit is supplied in a compact, durable, vacuum-formed book specially designed to contain the items required for basic maintenance of a Commodore microcomputer system.

The products included in the kit include;

SAFECLENE

– a tape-drive cleaning fluid with a safe solvent to clean all delicate surfaces such as tape heads.

FOAMCLENE

– an anti-static foam cleanser which lifts grease, dust and dirt from keyboards, plastic cases and covers and all surfaces including fabrics.

SAFEWIPES

– pads that are soft, lint-free, absorbant and excellent for cleaning all delicate external surfaces.

SAFECLENS

– anti-static screen wipes specially designed to clean VDU screens, reducing the risk of errors and eye strain caused by imperfect visual display.

SAFECLOTHS

– spun-bonded cleaning cloths that may be required for a stronger, more uniform wiping material.

FLOPPICLENE

– flexible diskette and disk head cleaners. These are recommended for cleaning all Commodore floppy disk units. A special jacket takes a cleaning pad which is inserted into the drive, eliminatiang the risk of head recontamination and abrasion.

To ensure maximum reliability of your Commodore system, and to ensure your investment is always clean and presentable, ask your Commodore Dealer for more information on the PETkit products.

AID4U UPDATE

The AID4U which was reviewed in issue 2 Volume 2 of the Commodore Magazine is now available for both the 4000 and the 8000 series computers.

The chip contains an updated version of both the programmers tool kit, BASIC Aid and as such offers a whole range of editing and program development commands. In total the AID4U EPROM adds 26 commands to the basic machine.

This chip is now available for $55.00 retail; which is a fair price considering the number of commands the package contains. Remember, as the chip is able to 'sit' in either spare ROM socket please order the desired version $9000 or $A000 as well as machine model.

For details call:-

Yarralumla Software
141 Bailey Place
Yarralumla NSW 2600
Ph (062) 82 1379

NEW VISICALC PACKAGE

A new VISICALC package that has recently been introduced, contains the correct programs for all 32K plus Commodore computers.

The package is supplied with two diskettes, CBM 4040 and CBM 8050 format. The starter program will determine which program to load to suit the host computer.

The manual supplied is a detailed training and reference guide to using the powerful VISICALC program.

COMMODORE ACCOUNTING MANUALS

These manuals have been professional written in a bound book format to give a detailed training and reference guide to the Commodore Accounting System.

Volume 1 contains detail on the Commodore Debtors and Stock System, and Volume 2 details the Creditors and General Ledger System.

Each manual contains detailed flowcharting showing how to implement these accounting packages into an office environment with the necessary control and audit documentation.

Detail has been provided to enable these manuals to be used as a class set in the instruction of computerised accounting systems and gives worked examples of a company's trading over a twelve month period.

Further details can be provided by your Commodore Dealer.

COMMODORE COURIER

This excellent 20 page newspaper has been published by Commodore Australia to give you all the latest information on Commodore products and pricing.

It is a summary with specifications of all products currently available, with reviews on products from suppliers other than Commodore.

If you would like a free copy of this newspaper, please contact your Commodore Dealer or write to Commodore Information Center.

COMMODORE COURIER

OFFICIAL PUBLICATION OF COMMODORE BUSINESS MACHINES

P.O. BOX 336, ARTARMON, NSW. 2064 PH: (02) 437 6296 SUMMER 1982

COMMODORE POISED TO REPLACE APPLE II

New computer announced

Commodore recently announced a new addition to their range of microcomputers and again Commodore has broken the price/performance barrier. Called the Commodore 64, this new personal computer offers many new and advanced features at a fraction of the cost of competitive computers.
The Apple 11 personal computer is the closest rival to the Commodore 64, but when price and features are compared, Commodore is confident the Commodore 64 will replace the Apple 11 as an alternative to users wanting the features of colour, high resolution graphics and sound, all at a realistic price.

Commodore 64

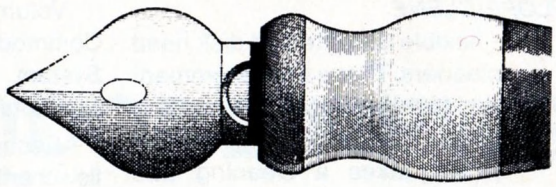
"THE COMMODORE 64 COULD BE THE MICRO-COMPUTER INDUSTRY'S OUTSTANDING NEW PRODUCT INTRODUCTION SINCE THE BIRTH OF THIS INDUSTRY."

VIC-20 OUTSELLS OTHER MICROS

HARD DISK DRIVES INCREASE STORAGE CAPACITY

Providing increased storage capacity up to 9 megabytes, Commodore's recently announced hard disk drives extend the range of Commodore peripherals, to offer one of the largest and most flexible system configurations available from any micro-computer supplier.

Vol.2 Issue 4 Commodore Magazine 3



LETTERS TO THE EDITOR

Dear Sir,

I am writing with bouquets, brick-bats and a query.

Firstly the good news:

As a new micro user (though not new to computing) I am very impressed at the existence of a publication such as "Commodore Magazine". This magazine is an essential link between the manufacturer and the end user; providing new feature/product update, hints and tips, general information and sharing of ideas. Commodore Magazine obviously achieves this, and I congratulate you – keep up the good work!

However, I cannot understand, what appears to be, a lack of attention to the checking and proof-reading of material published. I have only received Issues 1 and 2 (Vol. 2), but both contain glaring examples of missing (or wrongly placed) paragraphs and incomplete program listings. It was good to see that you reprinted and completed the "grappling with graphics" article – I hope you will do the same for the "PET POKEing" program listing. ("The pageing routine at line 2000 is handy and could be used to good effect in other programs" – if only we could read it in full!)

My query relates to back issues; I would like to receive all Vol. 1 back issues – what will this cost me.

Yours sincerely,
John Morter.

Thank you for your letter. We appologise for the errors in laying out the magazine. That is the human element of the magazine production, as all the keyboard entry and typesetting is done by computer.

We will republish Tony Ellis' program PET POKEing with a complete listing in a future issue of the magazine.

All back issues from Volume 1 are available except for Issue 1. These can be ordered as a subscription for one year ($A30.00) or as single issues ($5.00 each). An index of the contents of Volume 1 was published in Volume 2 Issue 2. The BEST OF PET Book is supplied in lieu of Volume 1 Issue 1.

Dear Sir,

In Volume 2 Issue 2 there is an error in the code for OLD ROM PETS on page 35 of the CASSETTE MERGE article. Second line of code at the bottom of centre column should be: POKE 525,1 : POKE 527,13:.

There are also three errors in Dr Greg Perry's programs on pages 27 & 28. Also in Gregory Yob's article on page 25, but no doubt you print them as received.

In Vol. 1, Issue 2 on page 4 the answer to the first question states that a PEEK(65532) will return a zero for BASIC 1.0. I get 56, but for location 5000. I get zero. I have seen this location used for some programs.

In Vol. 1, Issue 1 you repinted a BASIC 2.0 RAM/ROM MAP from Jim Butterfield. Do you have one for BASIC 1.0?

In the same issue on page 19 in Butterfield's Cross-Reference program he states that CHR$(0) is a null string. This is not true for BASIC 1.0 because I get LEN(CHR$(0)) = 1. I find it difficult reading a CHR$(0), '@', from tape using GET#1, A$.

Apart from the minor printing errors I think that the magazine so far has done it's job in disseminating information. I would like to see more articles on machine-code programming. The articles on the IEEE BUS I found interesting.

I have built a joystick interface and a programmable sound generator and rhythm box. I converted the COMPUTED GOTO, Vol. 1, Issue 2 page 20, to BASIC 1.0 which took some time as I had to build up part of the BASIC 1.0 RAM/ROM MAP.

Yours faithfully,
William Sands.

Thank you for your comments. We will not be publishing a memory map of BASIC 1.0 but this is available in the BEST of PET book available from all Commodore Dealers at a recommended price of $20.00. The product code is 512010. A complimentary copy has been sent to you.

BOOK REVIEW

We present here an objective review of one of Commodore's many publications, by a freelance expert in the field.

VIC 20 Programmer's Reference Guide

by Robert Baker

Reprinted with permission from **Microcomputing**, June 1982

As mentioned in early announcements, the book is divided into four sections: Basic Programming, Programming Tips, Machine Language and Input/Output.

A short applications guide is really a bit of subtle advertising for various VIC accessories and programs, but it does give a nice list of ideas on ways to use the system. Besides the normal reading material, the book has a number of useful charts and tables in the appendices. For hardware enthusiasts, there's even a full schematic of the VIC 20 inside the back cover.

The first part of the book describes the various commands and operations of VIC BASIC in detail. It's a handy yet thorough reference for VIC BASIC, but does not attempt to teach you how to actually program. Each entry in the BASIC vocabulary guide explains how the instruction is used and includes simple examples to help clarify matters. You'll even find information on how to abbreviate most of the commands to save typing time or to cram more commands into each program line. The sections on numbers, variables and operators should be especially helpful to newcomers in the world of computers.

The second portion of the book covers various programming tips for writing your own BASIC program. About one-third of this section covers cursor controls and program editing, using the GET statement, and simple discussions of various ways to save memory within the programs. The remaining two-thirds covers the use of graphics and sound, with a good deal of information packed into those 20-some pages. There's a nice description of the programmable characters and how you can even use them for high resolution or multi-color graphics. Several sample programs are included at the end to help illustrate the techniques covered, including the mixing of sound and graphics.

The third part of the book is an introduction to machine language programming and the internal workings of the machine. It attempts to provide information for all levels of users, but is primarily for the more advanced programmers. It starts out with an overall functional description of the VIC 20 to give you an idea of the way the VIC 20 processes programs within the system. The system overview contains a block diagram of the system as well as the internal 6502 microprocessor itself. Simple memory maps are included along with a discussion of how a BASIC program is stored in memory. All this information should be useful to some degree to just about any VIC user.

The last part of the book covers input and output to the VIC system. There's a complete description of the User Port, the Serial Bus and the VIC Expansion Port. There's a big write-up on the RS-232 interface, but a few important details are omitted. In particular, a previous section of the book refers you to this section for the valid secondary addresses when OPENing the RS-232 channel, but the information is just not there. It would have been really nice if there were some information on actually connecting RS-232 devices to the VIC for those unfamiliar with the RS-232 handshake lines. Brief information is also included in this last section on using a joystick, paddles or a light pen with the VIC. There's even a short section on the VIC graphic printer and how it's used.

There are a number of charts, maps and tables in the appendices, and most are very useful and handy references. However, novice programmers might need more help than what's presented in Appendix I when converting programs to VIC 20 BASIC from other systems. The authors only touch the surface with the information they present, but it should be useful. Don't forget, there's also a full VIC 20 schematic and a complete index as well.

As a whole, the book is very well done and probably the best I've seen from Commodore. It provides information of value for users at all levels of experience. As its name suggests the book is a reference guide for programmers. It will not teach you how to program, but it will provide a wealth of information in one handy source that is just not available elsewhere.

COMPUTERS JOIN THE WORK FORCE

Article prepared by Jenni Gyffyn, of Gippsland Computer Business Services – Phone (051) 52 5939

Most people involved in business – whether managers or employees – find that there are two parts to their work activities. One is the actual service they are providing, and the other is the paperwork and decisions to be made regarding that service they are giving to their customers. It is in that second part of a business that computers are helping out increasingly.

Small transport and parcel delivery companies can use a Transport System to take care of their delivery docket details, customer, depot and sub-contractors' files, money owing and statements, and monthly income reports.

BENEFITS:

— more time can be devoted to managing the business and attending to the needs of the clients.

Dairy farmers and vets have a Dairy Herd Management program that will keep records of calving and heat dates, number of times mated, production and disease reports on both a herd and individual cow basis.

BENEFITS:

— increased veterinary input into the dairy farm management with greater returns to the vet.

— aids the farmer in keeping complete records and predicting management requirements.

Car dealers throughout Australia use a Finance and Insurance system. Finance deals, finance/cash and finance/lease comparisons, quotes and so forth are instantly calculated from the computer.

BENEFITS:

— easier to provide the customer with financial assistance.

— paperwork is reduced which leaves time for the salesman to see more clients.

— increased finance deals which increases the salesman's and dealership's commissions.

Many businesses use VISICALC, a powerful planning and forcasting program. It eliminates those hours of using a calculator, pencil and rubber, and allows the boss to say "What if I take on an extra employee?" or "How would the business benefit if I...".

BENEFITS:

— budget planning, calculations and forecasting can be done in seconds, and the boss can play with many ideas involving money.

Public accountants, as well as making personal contact with their clients, have an awful lot of "hack-work" involving shunting figures around, coding and preparing reports. A Public Accountants package run on their computer will carry out all the time-consuming tasks, as well as many other functions. To avoid detailing these, particularly as most of it would make sense only to a qualified accountant, the benefits can be summed up as saving time for the accountant so he can spend more time seeing his clients and becoming more in touch with their business and its future, accuracy of figures and reports become easier to produce, and the client ends up paying less for this specialised service.

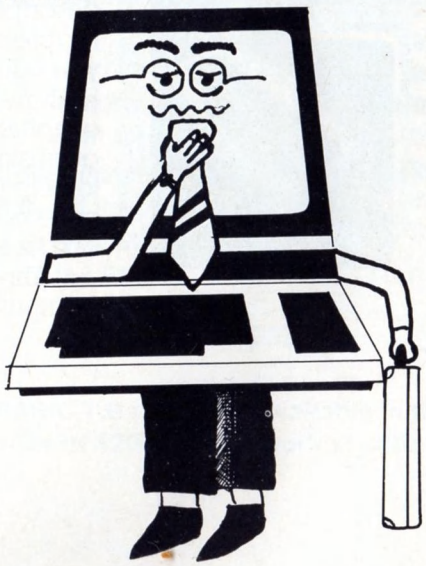
Nearly any business has the monthly task of dealing with Accounts Payable and Accounts Receivable. Stock control is another universal piece of house-keeping. As a business grows, extra staff are frequently hired to cope with these accounting functions. An alternative is to have your own computer sitting on a table in your office keeping track of all the money owed to you, printing statements, aging reports, reminder letters, customer lists, stock levels and re-order reports, profit/loss reports, who you owe money to, cash forecasting (so you know how much you pay each month to keep within trading times), and all those other jobs that create paper warfare each week/fortnight/ month.

It is an interesting fact that not only can a business keep accurate and immediate control of what is on the shelves, where the money is going to and where the money is coming from, but the monthly lease on a computer costs far less than an employee's monthly wage. The abovementioned systems cost between $2,500 (small car dealer's system) to about $12,000 (Public Accounting system). If you look at the more expensive computer as costing about $535 a month to lease you see how feasible this solution can be.

This is not to say that staff should be reduced, but it does mean that existing staff can be employed in direct production without the frustration of them being diverted into paperwork -let's face it, paperwork can kill a business! Production doesn't.

Another alternative is to hire a computer bureau to handle those functions that can be dealt with externally to the business: That is, debtors, creditors, profit and loss reports etc.

A number of microcomputers, including the Commodore Business Machines, provide all the above services.



EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT COMMODORE COMPUTERS *

* And Asked!

PET 2000/4000 SERIES

1. **Q: Can the memory of the 4000 series computers be expanded beyond 32K?**

   A: The memory of the 4000 series computers can be expanded provided the machine has a universal board and 32K RAM. The 64K expansion board will provide 96K of RAM.

2. **Q: Where can a conversion kit for the keyboard of the original PET 2001 computer be found?**

   A: The original flat keyboard of the 2001 series can be converted to a regular typewriter keypad by using keyboard interfaces manufactured by Skyles Electric Works of Mountain View, Califonia.

3. **Q: How can I tell which set of ROMs my PET 2001 computer has?**

   A: Upon power-up, the BASIC message that appears on the screen will indicate which set of ROMs the PET 2001 has. If a "*" sign appears, it is original BASIC 1.0 machine. If a "#" sign appears, it is a BASIC 2.0 or higher machine.

4. **Q: How many cassette recorders can be connected to the 4016 computer?**

   A: The 4016 has two cassette ports, which enables you to connect two cassette recorders at one time.

5. **Q: Can WordPro 4 be used with the PET 2001 computer?**

   A: No, the PET 2001 is a 40-column machine and WordPro is designed to work with an 80-column machine. However, WordPro 1, 2, and 3 are all 40-column versions and can be used with the PET 2001 computer.

6. **Q: Which disk drive is recommended for use with the PET 2001 computer?**

   A: All Commodore disk drives work equally well with the PET 2001. Customers should choose the one that fills their storage requirements and budget.

7. **Q: With respect to the PET 2001 computer and the 2040 disk drive, how can a record be deleted?**

   A: To delete a record on file, create a new file (possibly with the same name) and write all of the records to it – with the exception of the one to be deleted – then scratch the original file.

8. **Q: Can RS232C devices be connected to the computer?**

   A: Yes. There are a few IEEE to RS232C conversion interfaces available. Your Commodore Dealer will be able to advise the best to suit your application.

CBM 8000 SERIES

1. **Q: What languages are available for the 8032?**

   A: The 8032 is capable of executing FORTH, PASCAL, LISP, COMAL, and PILOT, as well as Microsoft BASIC and 6502 Assembler. with the 64K Memory Expansion Board, the 8032 is capable of executing UCSD PASCAL.

2. **Q: What is a memory map?**

   A: A memory map is a list of all the specialized memory addresses in the PET/CBM/VIC computers. A separate map is required for each version of BASIC. These have been published in recent copies of the Commodore Magazine.

3. **Q: Can the 8032 interface with a TV monitor?**

   A: The 8032 can interface with a monitor, but a high-resolution monitor (16 MHz or greater band width) is

4. **Q: Can files made in 6502 mode be read in 6809 mode and vice versa?**

   A: Yes, with software translation, files made in 6502 mode can be read in 6809 mode and vice versa. The translation needed for 6502 mode is PET ASCII, and the translation needed for 6809 mode is U.S.ASCII.

5. **Q: Where is the cursor address stored in the 8032?**

   A: One the 8032, the column position of the cursor is decimal address 198. The row position is decimal address 216.

6. **Q: How can three portions of one program be merged together when using the 8032 and the 8050 disk drive?**

   A: Programs such as the "Programmer's Toolkit" allow merging of programs. Commando Chip and its merge command can be used, or programs can be merged with machine language programming.

7. **Q: How is "pi" accessed on the 8032?**

   A: To access "pi" on the 8032, PRINT a CHR$(255) while in the graphics mode. If you are in upper case/graphics mode, POKE 32768,94 and this will put a "pi" in the upperleft corner of the screen.

8. **Q: How fast does the 8032 transfer data to the 8050 disk drive?**

   A: The rate of the IEEE bus is approximately 9 kilobytes per second.

9. **Q: Can the character format be changed on the 8032?**

   A: The character set may be changed by programming your own EPROM in place of the character generator ROM.

SuperPET

1. **Q: When using the modem with the SuperPET, why do characters sometimes seem to be echoed back onto the screen?**

   A:The built-in communications program on the SuperPET prints the characters on the screen as they are keyed in from the keyboard. If the host computer is set to echo, then the received character will be a duplicate, causing the screen to display each character twice. Almost every host computer can be made to not echo. That command depends on the host computer being used.

2. **Q: Does the SuperPET have double precision?**

A: Double precision implies twice the normal number of digits of accuracy. The SuperPET allows for nine digits of accuracy for all operations, which is better than single precision on other systems.

3. **Q: Is there a compiler being developed for the SuperPET?**

A: At this time, compilers are being developed for all the SuperPET languages written by Waterloo.

4. **Q: What pins are required to connect a modem to the SuperPET?**

A: Pins 2, 3, 5, 6, 7, 8, and 20 are required to connect a modem to the SuperPET. A straight-through 25 pin RS232 modem cable works very well and is inexpensive.

5. **Q: How can the screen be cleared from within the program on the SuperPET?**

A: The screen cannot be cleared from within the program. However, an alternative is to print 25 blank lines, which will cause the text to scroll off the screen.

6. **Q: Is all of the RAM accessible for programming on the SuperPET?**

A: Assembler Language programs can access all 96K of RAM in the computer. The higher level languages load the interpreter into upper 64K of RAM and leave the lower 32K of RAM for programming.

7. **Q: What type of communication signal does the SuperPET send?**

A: The SuperPET will receive and transmit a standard RS232 signal at 300 to 9600 baud over the built-in RS232 port. Of course, the SuperPET still has the IEEE port and the User port.

8. **Q: What ASCII is used with the 6809 and the 6502 microprocessors?**

A: the 6809 microprocessor utilizes standard ASCII (American Standard Code for Information Interchange), while the 6502 microprocessor uses a variation called PET ASCII.

9. **Q: How much memory is available for programming in the various SuperPET languages?**

A: All the SuperPET interpreters (except APL) leave approximately 30 kilobytes available for programming. APL provides a workspace of approximately 28K. Using 6809 or 6502 Assembler programs, the entire 96K is available.

10. **Q: What functions will the COBOL language interpreter include?**

A: The Waterloo microCOBOL is a subset of ANSI-1974 Standard COBOL. All of the Level 1 features including the NUCLEUS, SEQUENTIAL I/O, RELATIVE I/O, and TABLE-HANDLING are supported. Certain features of Level 2 are also supported including the PERFORM, STRING, UNSTRING, and INSPECT verbs.

VIC 20 PERSONAL COMPUTER

1. **Q: Is there a Modem for the VIC-20 computer?**

A: Yes. If you wait for the review in the Commodore Magazine, we will provide all the details.

2. **Q: How is bit mapping accomplished on the VIC 20?**

A: Bit mapping is accomplished by putting consecutive characters on the screen, then re-programming the dots of the characters. This concept is elaborated in the Programmer's Reference Guide.

3. **Q: How are "xy" graphics plotted on the VIC 20?**

A: To plot low-resolution graphics, use the cursor control characters imbedded in PRINT statements. For high-resolution plotting, the VIC Super Expander cartridge is recommended.

4. **Q: How is the clock displayed on the VIC 20?**

A: To display the clock on the VIC 20, PRINT TI$ by PRINTing TI$="033000". This means 3:30 and no seconds.

5. **Q: Is PASCAL available for the VIC 20?**

A: PASCAL is available for the VIC 20 but has not been released in this part of the world as yet.

6. **Q: Is there a way to determine how much memory is left in the VIC after RUNning a program?**

A: You can determine how much memory is left after RUNning a program by typing: PRINT FRE(0)

7. **Q: Do the various application software cartridges reduce the amount of RAM available?**

A: The Programmer's Aid and VICMON Machine Languages Monitor cartridges do not reduce the amount of RAM accessible to BASIC. The VIC 20 Super Expander adds 3 kilobytes to RAM.

8. **Q: What telecommunication networks work with the VIC 20?**

A: The VICModem allows the VIC to access such networks as The Source, Compuserv, Micronet, Dow Jones, and the New York Times, and locally The Australian Beginning.

9. **Q: How many digits does the VIC floating point have?**

A: The floating point variable routines in the VIC have nine significant digits for the mantissa, and the exponent in the range of -38 to +37.

10. **Q: What is the difference in the voltage levels of the VIC's RS232 and standard RS232?**

A: The VIC voltages are at TTL level (0 to 5 volts), while the RS232 standard is -12 to +12 volts. In addition, the signal levels from the VIC are inverted from the standard RS232.

11. **Q: Is it possible to hook a standard audio cassette recorder to the VIC 20?**

A: No. A VIC Datassette is necessary due to the special recording method used to obtain the high reliability. You can use standard audio cassettes. For best results use a short tape (C10 or C15), but do not use chromium dioxide tapes as these cut the high frequencies which is required to store the data.

12. **Q: What version of BASIC is used in the VIC 20?**

A: The VIC-20 uses PET BASIC 2.0.

13. **Q: How can I obtain VIC schematics and VIC memory maps?**

A: These are contained in the VIC-20 Programmers Reference Guide which is now available from your Commodore Dealer.

DISK DRIVES

1. **Q: Is it possible to use hard-sectored diskettes with the 4040 disk drive?**

A: Yes, it is possible to use hard-sectored diskettes with a 4040 disk drive because it has no sector-hole detect sensor. The 4040 disk drive ignores the sector holes in the diskette so any kind will work.

2. **Q: How can relative files be copied from a 4040 to an 8050 disk drive?**

A: To copy relative files from a 4040 to an 8050 disk drive, the side sector blocks have to be copied. This can be done with the "Copy All" program which is on the diskette supplied with all drives.

3. **Q: Can several PET/CBM computers be connected to a disk drive?**

A: Although it is not recommended, it is possible to cable together more than one computer to a disk drive. Be careful to have only one computer accesses the disk drive at any one moment, otherwise the system may crash. One way to prevent this problem is to use the MUPET, which is sold by Canadian Micro Distributors of Milton, Ontario, or with a networking system such as the Hydra. Another way is to use the REGENT, which is available in Australia. This device allows connection of up to sixteen computers to one drive and it works especially well in a school environment.

4. **Q: Can more than one disk drive be connected to a computer?**

A: Up to eight disk drive units can be connected to a single computer. It is possible to set the unit address on each drive as a device number between "8" and "15". This can be done under program control or as a hardware modification.

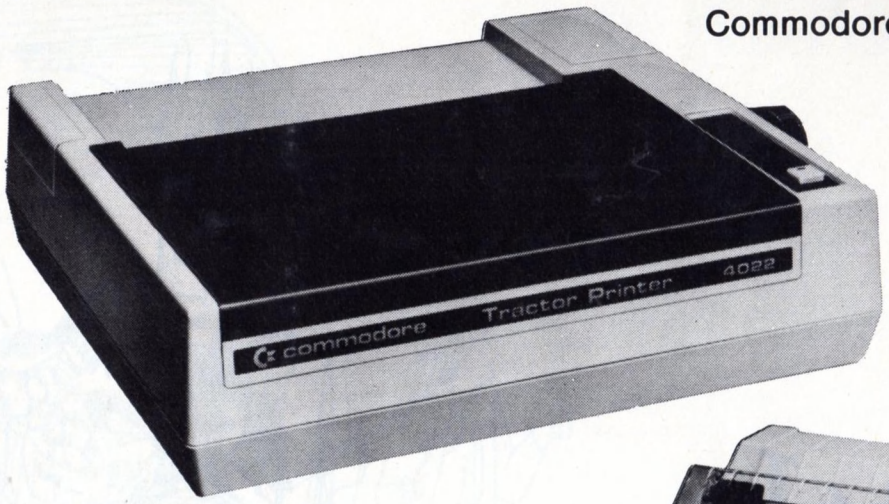
5. **Q: Where can an explanation of random, sequential, and relative files be found?**

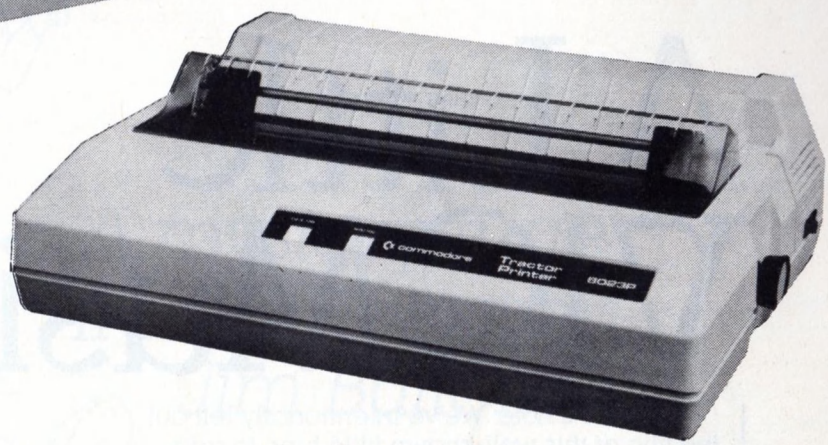
A: For a detailed explanation of these files, refer to the PET/CBM Personal Computer Guide (Second Edition). This book is available from your authorized Commodore Dealer.

6. **Q: Is it necessary to load the DOS into the disk drive?**

A: The DOS (Disk Operating System) is included in ROM on the main logic board of the disk drive. Therefore, it is not necessary to load the DOS – the drive is ready when it is powered on.

Commodore Tractor Printer 4022





Commodore Tractor Printer 8023P

PRINTERS

1. **Q: Do Commodore printers require a unique type of paper?**

   A: Most Commodore printers are capable of using ordinary bond paper. Paper for all printers is available from authorized Commodore Dealers.

2. **Q: Can legal-size paper be used with 8023 printer?**

   A: Standard 11" x 17" computer paper can be used with 8023 printer.

3. **Q: Can the 4022 printer handle mixed formatted data?**

   A: The 4022 printer can handle both numeric and alpha characters.

4. **Q: What printers are compatible with the 8032?**

   A: All Commodore printers are compatible with the 8032. By using a PET to serial interface, some ASCII printers will also work.

5. **Q: Where can necessary supplies for printers be obtained?**

   A: Most authorized Commodore dealers stock ribbons, paper, and other supplies for use with Commodore printer.

6. **Q: Does Commodore manufacture interfaces to allow the use of non-Commodore printers?**

   A: There are vendors that manufacture interfaces to allow the use of some non-Commodore peripherals. However, Commodore peripherals do not require interfaces for use with Commodore computers.

7. **Q: How can a program be listed upper/lower case on the 8023 printer?**

   A: Set the printer to upper/lower case by issuing this sequence of commands:

   OPEN 1,4,7
   PRINT#1
   CLOSE!

   The printer will now print in upper/lower case.

8. **Q: What are the misprints in the 4022 printer manual?**

   A: On page 27, the last line in the example should read:
   PRINT#8, CHR$(20).
   On page 31, the second line in the first example should read:
   PRINT# 6, CHR$(195)
   This value will produce one line per inch. The proper value to use for six lines per inch is 36.

9. **Q: Can the printer be made to print a certain number of lines per page?**

   A: This technique is called "paging" and can be utilized on the 4022 and the 8023 printers. The command to turn this feature on is:

   OPEN 1,4
   PRINT#1, CHR$(147)

   This command sets the number of lines per page to 66. To set a different number of lines per page, for example, 40, follow the first command with:

   OPEN 2,4,3
   PRINT#2, CHR$(40)

   The number in the parenthesis indicates the number of lines per page desired. To turn paging off, type:
   PRINT#1, CHR$(19).

A Little VIC Music

(Editor's note: We've intentionally left out the title of this well-known little tune to add an element of mystery and surprise to your endeavors. Forgive us, Jim.)

The following program plays music on the VIC. The music is listenable, and the program is worth looking at, too.

You'll note that the three voices of VIC are different. Voice three is sharper, and is better for carrying the tune. Voice one is the softest.

Hope you don't mind my breaking up the listing with comments.

```
 90 REM: VIC MUSIC/JIM BUTTERFIELD/DECEMBER 81
         This tells you who to blame
100 DIM A(8)
```

Makes room for eight voices. How come? We only have three voices on the VIC and four "lines" in the song. Watch for the trick.

```
110 POKE 36878, 15
```

Set the volume to maximum.

```
120 FOR A = 5 TO 0 STEP -1
```

Here's our main loop. We're going to play the tune six times.

```
130 T = TI + S
140 IF TI < T GOTO 140
```

This waits for time "s" before allowing the program to continue. The time is measured in "jiffies": units of 1/60 second.

```
150 READ S, A(A + 0), A(A + 1), A(A + 2), A(A + 3)
```

Here comes the song data. It's taken from the DATA statements near the end of this program. We're reading the data into the table cleverly; this way, each voice "comes in" at the proper time.

```
160 POKE 36874, A(3): POKE 36875, A(4): POKE 36876, A(5)
```

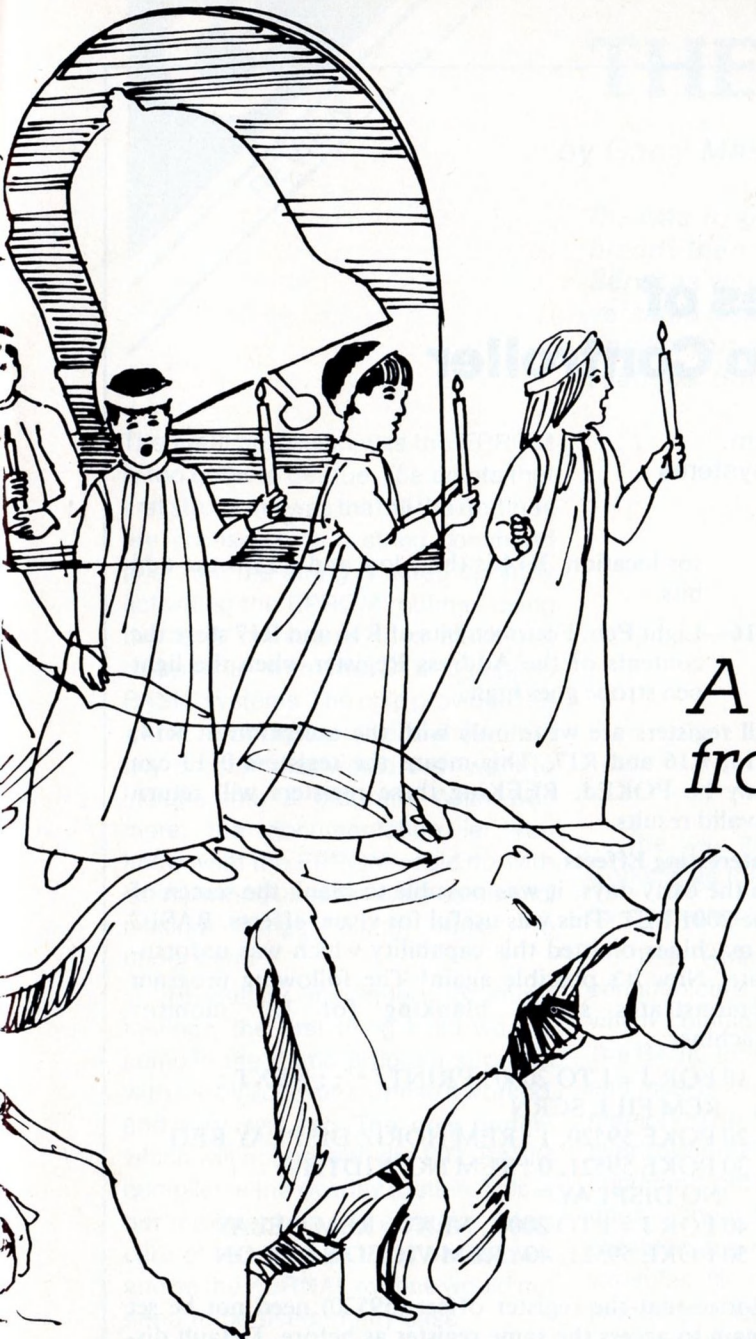
Play the music! This puts the notes into the VICs playing electronics.

```
170 IF S <> 0 GOTO 130
```

If there's no more music to play, variable S will become zero (from the data statement at line 1120) We may want to do it again, though.

```
180 RESTORE: NEXT A
```

RESTORE takes us back to the start of the data statements (line 1000) so that we can play it again if we wish. NEXT A takes us back for the six repeats.



A mystery tune from VIC expert Jim Butterfield.

190 POKE 36878, 0: end
Turn down the volume and quit. The END statement isn't really needed here, but it's good practice.

The rest of the program is our DATA statements containing the music. It's set up with a timing value followed by the four "parts". By careful reading of the program, you may be able to work out how the different voices come in during the repeats (hint: the key to the trick is in lines 150 and 160).

```
1000  DATA 10, 195, 207, 215, 195
1010  DATA 10, 195, 207, 219, 195
1020  DATA 10, 201, 209, 215, 175
1030  DATA 10, 201, 209, 209, 175
1040  DATA 20, 207, 215, 207, 195
1050  DATA 20, 195, 215, 195, 0
1060  DATA 10, 195, 207, 215, 195
1070  DATA 10, 195, 207, 219, 195
1080  DATA 10, 201, 209, 215, 175
1090  DATA 10, 201, 209, 209, 175
1100  DATA 20, 207, 215, 207, 195
1110  DATA 20, 195, 215, 195, 195
1120  DATA 0, 0, 0, 0, 0
```

It's not very big, but it's interesting to see how the coding comes together. Check Appendix F of your VIC-20 Friendly Computer Guide and you'll see how to set up the notes. Write your own music. If you like programming you might want to try your hand at writing a program which allows DATA statements to be written in easier form. For example, line 1000 might be written as DATA 10,C,E,G,C . . . but your program will need to be smart enough to catch the letters and translate them into the appropriate numbers.

Music doesn't have to stand by itself, of course. You could add it as an extra touch to games and animations. Looking at it the other way, you could add to the music—how about a "bouncing ball" program that lets you sing along with VIC?

You can get some nice effects from the VIC, although you'll never quite achieve orchestra quality sound. I can recall showing a group of users some simple music coding on the VIC. At one point, I played a simple rendition of "Dixie", and noticed a listener who had tears in his eyes. I was touched. I asked him, "Are you a Southerner?"

"No," he replied. "I'm a musician."

I guess you can't win 'em all.

Features of the 6845 Video Controller

by
Jim Holtom
Control Microsystems

All Commodore machines that have 12″ monitors (8032, fat 4032, 8096, and SuperPET) employ a device known as the 6845 video controller to generate the video signals. Some come with a 6545, which is the same as the 6845.

The chip has 18 programmable registers that are accessed through 2 memory locations at $E880 and $E881 (59520 and 59521). To read or write a register, the number of the register (0–17) is stored in 59520 and that register can then be accessed through 59521. This method saves considerably on address space.

The device is used to control such functions as character height, cursor size, horizontal position, vertical resolution, etc. Here is a summary of the registers and their functions:

R0—Horizontal Total Register. Horizontal frequency equalling the total of displayed plus non-displayed "character time units" minus 1.

R1—Horizontal Displayed Register. Number of displayed characters per horizontal line.

R2—Horizontal Sync Position. Controls horizontal positioning.

R3—Horizontal Sync Width. Four bits which control the width of the horizontal sync pulse.

R4—Vertical Total. The vertical frequency is controlled by R4 and R5.

R6—Vertical Displayed. Number of displayed character rows on the video.

R7—Vertical Sync Position. Controls vertical positioning.

R8—Interlace Mode. 2 bits which determine whether interlaced or non-interlaced mode is employed.

R9—Maximum Scan Line Address. Five bits determining the number of scan lines per character row including spaces.

R10—Cursor Start. Seven bits for cursor start scan line and blink rate.

R11—Cursor End. Five bits for cursor end scan line.

R12—Start Address. R12 and R13 control the first address put out as a refresh address after vertical blanking. R12 is the low 8 bits and R13 is the hi 6 bits of the address.

R14—Cursor Register. This 14 bit register stores the cursor location. R14 is the 8 low and R15 is the 6 hi bits.

R16—Light Pen. Fourteen bits of R16 and R17 store the contents of the Address Register when the light pen strobe goes high.

All registers are write only with the exception of R14, R15, R16 and R17. This means the registers 0–13 can only be POKEd. PEEKing these registers will return invalid results.

Interesting Effects

In the early days, it was possible to blank the screen of the 2001 PET. This was useful for visual effects. BASIC 2 machines omitted this capability which was unfortunate. Now it's possible again! The following program demonstrates screen blanking for 12″ monitor machines:

```
10 FOR J = 1 TO 2000 : PRINT "*"; : NEXT :
   REM FILL SCRN
20 POKE 59520, 1 : REM HORIZ DISPLAY REG
30 POKE 59521, 0 : REM NO WIDTH =
   NO DISPLAY
40 FOR J = 1 TO 2000 : NEXT : REM DELAY
50 POKE 59521, 40 : REM VIDEO BACK ON
```

Notice that the register offset (59520) need not be set again to access the same register as before. Default display width for 40 and 80 columns screens is "40".

The horizontal and vertical positioning of the display area can be altered by POKEing different values into R2 and R7. R2 default is 41 and R7 default is 29.

The number of displayable character columns may be modified by POKEing different values into R1. R1 default is 40. Likewise, the number of displayable character rows can be changed using R6. R6 default is 25.

The machine makes use of the controller for things like windowing, and text/graphic modes. Several other effects could certainly be achieved for use in games and other applications. The combinations and permutations are virtually endless!

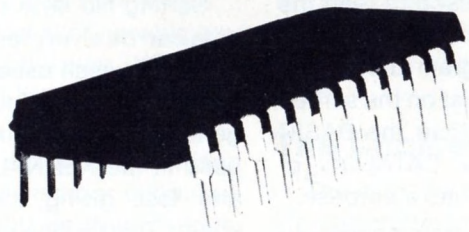
NOTE:

A word of caution. Uneducated experimentation with the 6845/6545 can potentially crash the computer. The effects involving the registers discussed are all safe to play with but other registers should be left alone unless you know what you're doing.

THE KERNEL.

by Garry Mason

If I was to give my award of 1982 for the 'best idea since sliced bread', then I'm sure that the 'KERNAL' from Gippsland Computer Services would be in the top three. This package is a number of very versatile routines which have been 'burnt' into an EPROM. It is not a Toolkit, but a set of data entry and very useful sort and interactive routines. But, very importantly, it is designed to be used at run time.



The first impression was the EPROM is too good to be true. The points that first struck me was that all the routines are accessed via a string command and then the string is acted upon by activating the EPROM routines using the SYS command, thus making it compatible with the DTL Compiled BASIC systems. The chip provides the whole set of interactive routines that any programmer would ever want to include into a program. What was more, the documentation let you know what the EPROM could do, with examples included for every command, making things much easier to understand.

To satisfy my 'need to know' feelings, the first thing I did was to compile the demo program supplied with the chip. To my surprise it worked, and very well too. The only routine which will not go with the DTL BASIC compiler is the overlay routine. This is not important as the compiler takes care of this type of thing on its own, and so the KERNAL routine would not need to be used in any case.

The one and only bug I found in the chip was it's inability to detect any syntax errors. Syntax errors either resulted in the chip using incorrect or default values, or just sitting in the middle of the CPU doing nothing.

The data input routines are very complex in that they are very flexible and cover every conceivable instance where data is required by the program. Basically there are two types of input available; those that paint the screen in reverse field thus showing the length of the input required, and those which leave the screen as it is. Again this is all that is required. Within this framework of the commands ('I' or 'R' -indicating reverse input) there are a number of parameters that need to be given, including the length of the field required and position on screen.

The way in which the command works is as follows:

```
G$="I[format][col]:[row]:[type]:[length]":sys40960
```

The variable G$ is used to communicate with the chip, telling it which command you want it to use. The BASIC interpreter takes the string and inserts it into the correct place in memory. The the 'SYS40960' hands over to the KERNAL which takes the command out of the string and off it goes. Once the result has been obtained, the results are put into the variables I$, r1, r2, r3 and r4. The parameters for the above command indicate the following:

I

– indicates the command – INPUT.

[format]

– this indicates how you want the information handed back.

n = normal mode.

j = justify field, trailing blanks or cash format for numerics.

o = pad with leading zeros.

[col]

– this tells the routine where on the screen in terms of columns and rows it should put the input.

[type]

– this indicates what type of characters are allowed to be inputted from the keyboard.

0 = all alpha/numeric characters.

1 = only numeric and decimal point, etc.

2 = all uppercase and numerics.

3 = only the first character to be forced into uppercase, rest all alpha numeric characters.

4 = date format in which all input is converted to the DD-MMM-YY format.

As you can see everything has been thought of, including the RUN/STOP key – which does not operate while the KERNAL has control. This means that any of the usual methods of disabling the RUN/STOP key still apply, for program protection.

The format for all the commands are handled in the sme way, that is, by using G$ to communicate the command. This means that it is possible to write a program that can build up a command by using the string handling commands LEFT$, RIGHT$, MID$ and concatenate. This is the way in which cursor positioning is done. The following example will flash an error message contained in DS$ in a position found in the variables X and Y.

```
G$="C"+STR$(X)+":"+STR$(Y):SYS4096:G$="E"+DS$
```

The "C" command stands for cursor locate. While the "E" command means flash error message at cursor position. Another type of printing routine is the prompt command "P" which is used to display a prompt before using the data entry routines.

These types of routines form the basis of any interactive program, they

are easy to use, fast and don't take up any extra memory space. This is a saving on both development time incurred at the time of writing and at run time, when the boot module and all the other extra codes needed to use machine code programs are loaded into RAM.

There are three other commands that interested me a great deal and these were the "N" or newly pack a variable, "U" unpack a variable and the disk fast get or "D". The disk read is a very remarkable utility, it's function is to read from a specified channel number a set amount of characters. This routine can be used to read the disk directory for example. This is what the demo program did – what speed, it was able to list on the screen the directory faster than the BASIC 4.0 disk commands CATALOG or DIRECTORY, which was a surprise.

The "N" and "U" commands are used for taking a number of strings of unfixed length and adding them together in a set manner. For example, we could use the "N" command to pack four variables of unfixed length into one string of forty characters long, thus adding together the first characters of each of the four strings. If one of the strings is less than ten characters long the "N" command will pad out the rest of the string to make it ten characters long. The "U" or unpack command does the same in reverse.

There are two main uses of this type of routine, either as a form of printer formatting when not using a CBM printer or for writing and unpacking a relative file record. This then provides a quick and straight forward way of using realtive files to the full. As the BASIC INPUT# statement can only read in 80 characters, the disk read utility comes into its own.

Sorting file keys or other types of data can be a very lengthly procedure in BASIC which usually means that a complicated machine code routine is generally used. The sort command used in the KERNAL is both flexible and fast, giving the emphasis on speed. This routine employs the Shell -Metzner technique which very basically compares elements half the array apart and the elements a quarter of the array apart and so on. This is compared to the normal 'ripple' or 'bubble' sorts, which sequentially compares it's way through an array.

This could work perfectly as an index method in conjunction with the "N", "U" and "D" commands. But I suppose someone has already thought of that.

So in conclusion a great little chip, worth the money just for the sort. The only thing that made me mad was it's inability to recover from syntax errors.

The recommended price of the KERNEL is $98.00 and further detailes can be obtained from;

GIPPSLAND COMPUTERS,
167 Princess Highway,
BAIRNSDALE 3875
ph (051) 525.939

INSTRUCTION SET.

CURSOR LOCATOR

g$="c[col]:[row]":sysy

DISK READ

g$="d[chan#]:[# chars]":sysy

FLASH

g$="e[message]":sysy

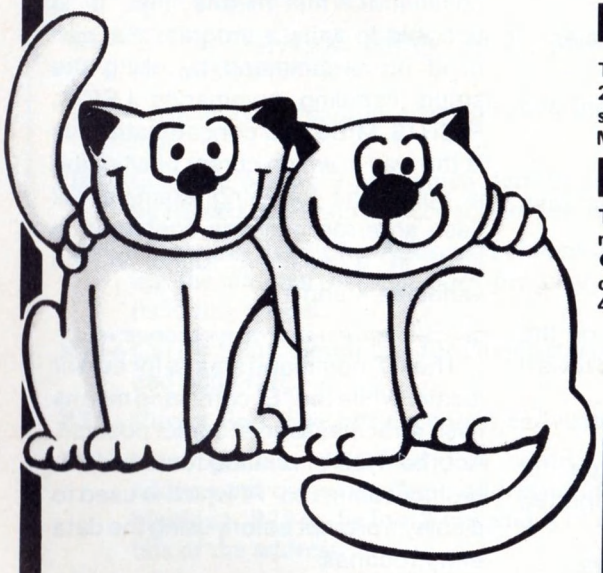
FIND STRING IN ARRAY

g$="f[array]:[skip chars]:[start#]:
[# search]:[string]:sysy

reply r2 – match pos r3 – # found r4 – # compares till larger

continued on page 18

Every PET needs a FRIEND...



Introducing a new series of programs that are 'FRIENDLY' to the user and represent outstanding value for money. . . .
and there will be more 'FRIENDS' coming

FRIEND 1 - Word Processor

An on-line ROM chip for 8, 16, and 32K machines. This Word Processor program has been written by professionals specially for The Microcomputer House.
The program can be used with or without a printer and will be available shortly in disk and tape versions as well.

WP CHIP	$85
DISK	$70
TAPE	$60

FRIEND 2 - Mailing List

This is a dual disk based system for the 4000 series microcomputers. It caters for 2,100 records per data disk and offers sort and select facilities. It will also be available shortly for single disk systems.
MAILING LIST $85

FRIEND 3 - Data Handler

This is a ROM chip containing a machine language routine which allows the programmer to control screen input. ● Alpha Field Entry ● Numeric Field Entry ● Date Entry ● Disk Fastget ● Field Reverse ● Field Flash. Each of these functions have different options. FRIEND 3 is a derivative of our security ROM used by all our packages.
4000 series and 8000 series $85 each.

All programs come with a complete instruction manual.

DEMONSTRATION STOCK AVAILABLE AT NEVER TO BE REPEATED PRICES
JUST PHONE OR CALL IN

Bankcard Mail Orders Welcome

The Microcomputer House Pty. Ltd.
1ST FLOOR, 133 REGENT STREET
CHIPPENDALE, N.S.W. 2008. PHONE: (02) 699 6769

COMPILER COMMENTS

By Jim Butterfield

I don't want to become involved in the Great Debate about compilers. On the other hand, it's almost irresistable to drive in and add a few footnotes. You'll find no product reviews here. Just a little talk about what's involved.

For BASIC?

Some languages were designed for compilers. In fact, the compiler was designed first, and whatever it turned out you had to type in ended up as the language. FORTRAN started more or less this way. To put compilers in perspective, we can do a little historical work.

Once, long ago, there were no interactive computers. You punched up a deck of cards and if you were lucky an operator would run them sometime that week. Most of the results came back saying something like SYNTAX ERROR (does that sound familiar?). There was no point in having an interpreter language; you wouldn't be there to watch it happen. We had FORTRAN and COBOL and others.

The first FORTRANs, for example, were tricky. If you used a variable called DIGIT, it would turn out to be a floating-point number; on the other hand a variable called NUMBER would be a fixed-point. Heaven help you if you typed TOTAL=TOTAL+1; you'd get a ?MIXED MODE error notice and have to recode TOTAL= TOTAL+1.0 to fix it. To input or output you needed to give more than the command: an extra line called FORMAT was needed, written in advanced gibberish. Honest.

Many of these problems have been corrected over the years – you did know that there was more than one FORTRAN, didn't you? – but style remains. The programmers have to adapt to the machine, and interactive is still an alien concept.

And Now, BASIC . . .

Along came BASIC. It's a loose language: you don't have to dimension some arrays; strings wander all over; sometimes you can have FOR and NEXT items that don't match (bad practice, but it can be done) ...and interactive users love it.

What's the problem? Some things are not clearly defined by BASIC. Let's look at a few of them.

Strings may be the worst thing that a compiler has to deal with. BASIC doesn't tell the compiler how big any string is likely to be – ever. INPUT X$ gives no hint as to the size of string X$. The poor compiler has a grim choice: allow maximum space for all strings and waste a lot of memory; or bounce the strings around as they change. The first alternative costs you program size; you write this little program that says DIM A$(1000) and the compiler immediately reports OUT OF MEMORY since it tries to allocate 255000 bytes for the array. The second alternative costs you time; no matter what you call it, some sort of garbage collection will have to take place. And then people complain beause they expect compilers to produce fast fast code.

At a first glance we think that the whole object of compiling is to get speed. But we don't give the compiler enough information to work up a really fast program. It's obvious that FOR J=1 TO 10 can run faster if we treat J as an integer. Unfortunately, we're not allowed to code FOR J% so the compiler will have to figure it out for itself. And what will it do with FOR J=A TO B? Until A is computed, we cannot know if it's integer or not.

It's obvious to us. We wrote the program. But the dumb compiler can't read our minds; and BASIC doesn't give enough explicit information to do the job.

One last example. It's one of the annoying things about BASIC that we sometimes have to code things like GET#1, X$: IF X$=""THEN X$=CHR(0) mostly to cover failings in BASIC itself. If I were hand-coding into machine language, I could replace the whole thing with one instruction, because I know that Machine Language doesn't have the "fault" that's in BASIC. But a poor compiler can't know that. It sees the GET instruction and codes it ... and it must add to the coding to generate the BASIC "fault" if it wants to be compatible. Then it must proceed to the IF statement and work through the coding to fix that same fault.

The Choices.

The compiler design has a choice. He can code for 99% compatibility, tracking everything that BASIC does quite exactly (including the faults). In doing so, he'll create a package in which almost anything will compile successfully. But, the compiled machine langauge will be doing most of the things that BASIC does, and won't be much faster than BASIC.

On the other hand, the designer can ask the user to make changes to this program before compilation that will help the process. He may also have things that compile from BASIC in a non-standard manner. He may make arbitrary decisions on BASIC structures – all for LOOP variables will be fixed-point, for example. And the compiler may question the user during compilation: How large is string M$ likely to be? Can J be fixed-point? The user has to work harder, but the end product runs faster.

Either way, the compiled program is not likely to be smaller in size than its BASIC source. It's difficult to code 100 IF J<5THENPRINT"J IS";J in less than the 19 bytes that BASIC uses. And good compilers add extra arithmetic – fixed-point addition, for example – that takes up overhead space.

Why Compile?

It's your choice. If you have a program that runs for five hours, you will probably be delighted with a paltry four-to-one compiler speedup. If you want protection against listing, a

compiler will do a good job of instant confusion.

Don't lose perspective. A program that spends most of its time waiting for an operator or for a printer won't speed up much under compilation.

Machine Language Programmers will be happy to know that they are not yet obsolete. Compilers can do a useful job. But until they get the brains equivalent of a human's judgment, they won't replace hand coding.

Ed Note:

One compiler that is available from your Commodore Dealer is the DTL BASIC compiler that is compatabile with almost all BASIC code and increases execution speed by up to 4 times.

---

continued from page 16

CLOCK HANDLER

g=$$"1*[var]:...:[varN]:[array1]:...:[arrayN]*[dlt from incl:[to excl]:[line # to run]:[overlay name]*":sysy

MONEY FORMAT

NUMERIC STRING

g$="m:[source variable]:[new len]:sysy

NEWLY PACK VARIABLES TO I$

g$="ni:[tot len]:[varl]:[len 1]:...:[varN]:[len 1]":sysy

PRINT PROMPT

g$="p[col]:[row]:[message]":sysy

INPUT WITH REVERSE SCREEN

G$="r[format][col]:[row]:[type]:[length]"sysy

SORT STRING ARRAY

G$="s[array name]:[skip chars]:[start #]:[# sort]"sysy

TRUNCATE BLANKS

g$="t[var name]":sysy

UNPACK INTO VARIABLES

G$="u[source var]:[tot len]:[varl]:[len 1]...[varN]:[len 1]"sysy

Pittwater Computer

NOW AVAILABLE.....

The Electronic Cash Book on both the 4000 Series and the 8000 Series Commodore Computers.

This programme is designed to exactly emulate the hand-written Cash Book but has the added features of keeping a running bank balance; automatic deductions of periodical payments -full details of these are kept on file; reconciliation with the bank statements and a printed list of unreconciled cheques.

It has full budgetry figures; automatic searches via cheque number, payee or amount; also full reporting functions with transaction listings, dissection summaries, and detailed dissection listings. Depending upon the version, there is up to 39 incoming dissections and 59 out going dissections.

We also advise that the 8000 Series will flow directly into the IMS/COMMODORE General Ledger, thus making it the ideal package for small businesses.

The retail price is $400.00 plus tax, where applicable. Please contact your local dealer or Pittwater Computer Sales

22 CARTER RD., BROOKVALE (02) 939 6760

LOADING A VIC PROGRAM ON THE CBM RANGE.

by Garry Mason

On all CBM computers, BASIC text is stored from the Hexadecimal address 0400 and moves down to 7FFF Hex. This space is known as the BASIC working area and is reserved for the storage of BASIC variables, string storage and the actual storage of line numbers (BASIC text). A number of pointers are used to calculate where each of these various sections of memory are located for use by the BASIC interpreter.

The main points to these regions are as follows:-

BASIC 3.0	BASIC 4.0	VIC 20	Description
122-123	40-41	43-44	Pointer: Start of Basic Memory.
124-125	42-43	45-46	Pointer: End of Basic, start of variables.
126-127	44-45	47-48	variables: End of variables, start of arrays.
128-129	46-47	49-50	variables : End of arrays.

At power on and under normal program execution, the start of BASIC on the Commodore microcomputers or CBM's is always preset to 0401 Hex., (1025 decimal). This does not change in most cases as it is not used. The most noticeable exception is where text is moved down in memory to make way for machine code subroutines. The start of BASIC pointer is then moved down, thus providing a protected area for the machine code routine. This is done so that the BASIC interpreter does not over write the subroutine.

The VIC 20 has the same structure including all of the above pointers which are used in the same way as the CBM range. The VIC has a different memory map compared to those of the other CBM and PET range of microcomputers. This was done mainly to make it considerably easier for the user to expand and then use the extra facilities of sound, colour and that of programmable characters offered by the VIC 20. This means that depending on the available memory or the number of different expansion units the start of BASIC pointer will begin at a different location.

The VIC without any extra memory will start recording BASIC text at address 1001 hex., leaving the statutory 00 byte at 1000 Hex. This means that if we take a program that was recorded using this configuration either on cassette or on a disk, the program will load into the same section of memory. To the CBM and PET range this does not meet it's criteria for a BASIC program, as the first three bytes after 0400 (it's start of BASIC pointer) are still 00 bytes and have not been over written by the incoming program.

To correct this situation we need only move the present start of BASIC pointer to locate at the start of our program at 1001 Hex. This can be done by POKEing the high byte value for this pointer to 10 Hex. or 16 decimal as in:-

```
POKE 41,16:CLR
```

This rectifies only the high byte pointer. The low byte of this address still contains a 01 byte which we still require to be 01, so it is not neccesary for any change to be done to this part of the address. As our program should have been LOADed over the first three 00 bytes the remaining one 00 byte needs to be POKEed in position just before the start of BASIC (1000 Hex.). This is also simple:-

```
POKE 4096,0
```

On the VIC the start of BASIC can be in one of three locations:- 0400 (as on CBM's) 1000 or 1200. The reason for this movement is that the screen will move also into one of two locations, this is to provide the longest possible continuous section for BASIC text or user memory.

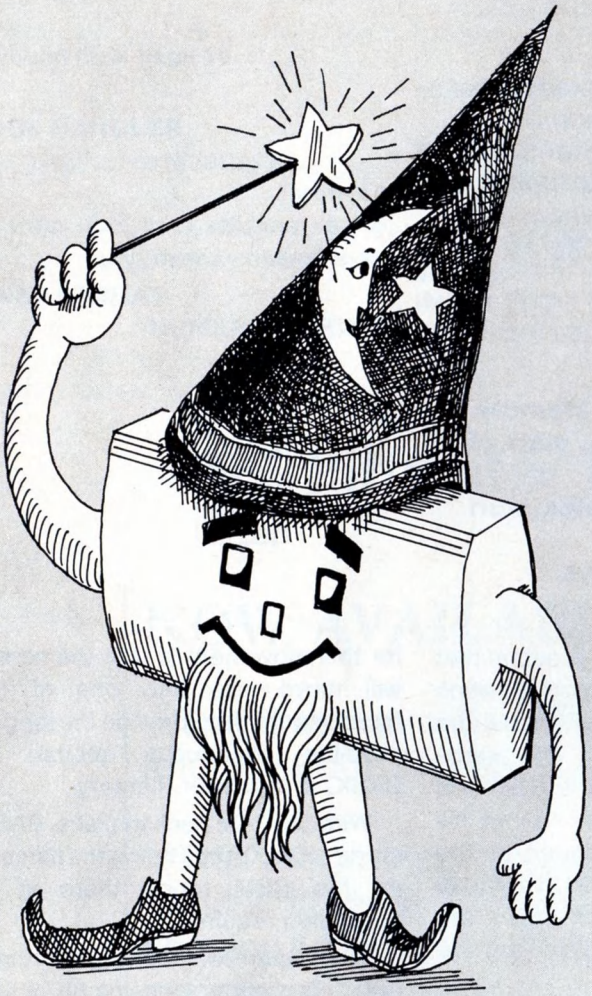
With 3K RAM pack in place, BASIC starts at 0400, thus being the same as on the CBM range there is no correction required.

With expansion memory starting at 2000 Hex., either by using 8K or 16K RAM packs or a mixture of the two cartridges, BASIC starts at 1200 Hex. and the memory mapped screen also moves to 1000 Hex. For which the correction is:-

```
POKE 4608,0 : POKE 41,18 : CLR
```

Using the VIC by itself, BASIC text is stored from 1000 Hex. onwards. The correction for this situation is:-

```
POKE 4096, 0 : POKE 41,16 : CLR
```



The VIC Magician
Using Those Mysterious Programmable Function Keys

Michael S. Tomczyk
Product Marketing Manager

Everyone always asks how the VIC-20's programmable function keys work (The function keys are those large yellow keys on the right side of the VIC-20 keyboard).

Special function keys were added to your VIC to let you take advantage of "one-key" programming features normally found on much more sophisticated (and expensive) office computers. It doesn't take much effort to program these keys and that's what the following lesson shows you.

When you first got your VIC-20, you were probably surprised that nothing happened when you pressed the function keys. That's because the function keys are **programmable,** meaning you have to program them to do something before they work.

Before getting into the programming part, let's take a closer look at the keys, themselves.

To begin with, the function keys are numbered 1 through 8. The odd-numbered keys 1, 3, 5, and 7 are obtained by simply typing those keys, and the even-numbered keys 2, 4, 6 and 8 are obtained by holding down the SHIFT key and then typing the appropriate key. This lets you use four keys to get 8 separate functions.

Now, here's an important point . . . each key has a special NUMBER which you must use when programming that key. This number is called the **CHR$ NUMBER.** All of the VIC's keys have CHR$ numbers, which are listed in the ASCII AND CHR$ CODES table on page 146 of your VIC user's guide (or page 273 of the VIC PROGRAMMER'S REFERENCE GUIDE).

CHR$ CODES FOR FUNCTION KEYS	
FUNCTION KEY NUMBER	**CHR$ CODE**
f1	CHR$ (133)
f2	CHR$ (137)
f3	CHR$ (134)
f4	CHR$ (138)
f5	CHR$ (135)
f6	CHR$ (139)
f7	CHR$ (136)
f8	CHR$ (140)

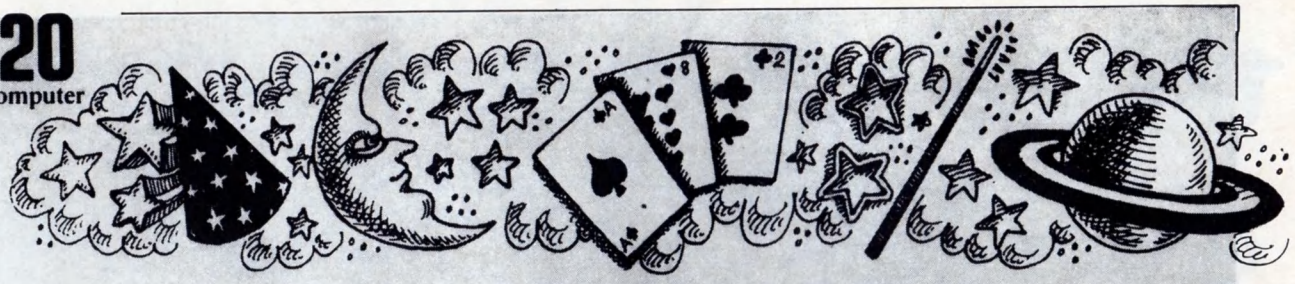
Note that the CHR$ numbers are not in exact order . . . the odd numbered keys are numbered 133–136 and the even numbered keys (which you must SHIFT to use) are numbered 137–140.

Using the GET Statement—"Hit any Key"

Before we explain how everything works, let's write a short program using a function key, to show you how it works. The first thing we do is TELL THE VIC TO LOOK AT THE KEYBOARD. This instructs the VIC to check the keyboard to see if you have pressed a key. The BASIC command for this is:

```
10 GET K$:IF K$ = "" THEN GOTO 10
```

This is called a "GET Statement" and is usually placed as a single line in your program. If you include this line all by itself in your program, the VIC will look for ANY KEY to be hit. So let's write a short program which tells the user to "HIT ANY KEY TO BEGIN." Enter this



program, type the word RUN and hit the RETURN key:

```
10 PRINT" HIT ANY KEY TO BEGIN":PRINT
20 GET K$:IF K$ = ""THENGOTO20
30 PRINT"■ R PROGRAM BEGINS."
```

(BEGINNERS: be sure to hit the RETURN key at the end of each line to enter it.)

THIS MEANS HOLD DOWN THE **SHIFT** KEY AND TYPE THE **CLR/HOME** KEY

THIS MEANS HOLD DOWN **CTRL** AND TYPE **RVS ON** KEY

See how it works? Now let's go one step farther. After we BEGIN our program, we may want to make the program WAIT until the user is ready to continue. This technique is often used in educational programs when you want to give a student time to study something on the screen before moving on. Hold down the RUN/STOP key and hit the RESTORE key to "exit" your program. Now type these additional lines (the VIC automatically adds them to the program above, which is still in VIC's memory):

```
40 PRINT"■ WAIT HERE.":PRINT:PRINT"HIT
   ANY KEY TO CONTINUE.":PRINT
50 GETK$:IFK$ = ""THENGOTO50
60 PRINT"■ R PROGRAM CONTINUES."
```

Note that this program now uses the same GET statement in two different places to check the keyboard to see if a key has been pressed. You can insert line 20 almost anywhere to make the VIC wait until a key is pressed . . . but if you do this in the middle of a program, don't forget to PRINT a little message telling the user to "hit any key to continue."

Now let's take a closer look at how the GET statement lets us check the keyboard . . . and program our function keys.

More Information About the GET Statement

As already explained, the **GET K$** line tells the VIC to check the keyboard to see if a key has been pressed. There are three important things to remember about using this line in your program. First . . . notice how we used the **string variable K$** in lines 20 and 50 above? Although we used K$ as our variable (K for "Key"), you can use **any legal string variable,** such as A$, KK$, B1$, etc. in this line. For example, you could use: **50 GETRR$:IFRR$ = ""THENGOTO50.**

The second thing to remember is that the GET statement line **always GOES BACK TO ITSELF.** In other words, if you put this line at line 100, the last part of the line would read . . . **THEN GOTO 100.** This makes the VIC keep checking over and over again until a key is actually pressed. So the GET statement includes a GOTO to its own line number.

Finally, don't attach any other BASIC commands on the same line as your GET statement. For example, don't put a colon and add more BASIC commands to lines 20 or 50 above.

Programming a Function Key

Now let's make a function key do something—how about amending our program above so that it only begins if you type function key 1, and only continues if you type function key 3 . . .

We'll use the SAME PROGRAM we used above and simply modify two lines and add a few extra lines to tell the VIC to accept ONLY DESIGNATED FUNCTION KEYS instead of "any key." To begin, type the word LIST and hit RETURN. This displays your program. Now modify your program by changing or adding the elements shown in BOLDFACE in the finished program below (If you need help editing program lines, see pg. 74 of the VIC-20 PROGRAMMER'S REFERENCE GUIDE, "Editing Lines.")

```
10 PRINT " HIT THE F1 KEY TO BEGIN":PRINT
20 GET K$:IF K$ = ""THENGOTO20
25 IFK$<>CHR$(133)THENGOTO20
30 PRINT" PROGRAM BEGINS
40 PRINT" WAIT HERE."PRINT:PRINT"HIT F5
   KEY TO CONTINUE.":PRINT
50 GETK$:IFK$ = ""THENGOTO50
55 IFK$ = CHR$ (135)THENGOTO60
56 GOTO40
60 PRINT"■ R PROGRAM CONTINUES."
```

LINE 25: Line 25 tells the VIC: "If you check the keyboard and find any other key but the F1 key being pressed, GOTO line 20 and check again. The <> sign means less or greater than (or you might interpret this as "not equal to") . . . in this case, the program treats CHR$ (133) as the number 133.

With this example, your "real" BASIC program would start at line 30. Naturally, you can alter (increase) the line numbers after line 30 to give yourself more room if you're inserting a longer BASIC program here.

CHR$ (133) is the CHR$ number for the F1 function key, and IF K$ = CHR$ (133) means "IF F1 IS BEING PRESSED." You can designate a different key by changing the CHR$ number (see chart above). For example, CHR$ (139) changes the key to F7.

LINE 26: This is IMPORTANT! In line 25 we told the VIC to GOTO line 30 if you press the F1 key . . . but we didn't tell the VIC what to do if we DON't press the F1 key. So what happens is this . . . the program will "fall through" the IF . . . THEN statement in line 25 and

KEEP GOING if we don't do something to stop it. The solution is to include the GOTO in line 26 which tells the VIC to GOTO line 10 and keep PRINTing that opening message until the F1 key is pressed. This is important because if you experiment with function keys you may find your program "falling through" the IF . . . THEN line and RUNning "wild." The answer is to insert a GOTO after the IF . . . THEN statement, which keeps the program going back to the prompt message until the proper key is pressed.

LINES 55–56: These lines are the same as Lines 25–26. Line 55 is the IF . . . THEN statement which designates the F5 key to be pressed and Line 56 keeps the program "looping back" to the message in line 40 until the F5 key is pressed.

Programming Function Keys to Perform Functions

So far, we've used function keys to 1) start your BASIC program, and 2) continue a BASIC program in progress. Both of these techniques used the function keys to add some nice cosmetic touches to your BASIC programming, but they really didn't show you how to use the VIC's function keys to perform "real" functions, so that's what we're going to show you next.

Type the word NEW and hit RETURN to erase your previous program, and type in the following program. This is a VIC SPEAKER DEMONSTRATION which lets you type any NOTE VALUE and hear it played on any of the VIC's four internal "speakers." Function key 1 plays the note you entered on Speaker 1 (the lowest tone speaker), function key 2 plays the note on Speaker 2, and so on. Speaker 4 is the "white noise" or "sound effects" speaker.

```
10 PRINT" ♥ SOUND DEMO":PRINT:
   POKE36878,15:S1=36874:
S2=36875:S3=36976:S4=36877
20 PRINT" ♥ "
30 PRINT"F1 PLAYS SPEAKER 1":PRINT
40 PRINT"F2 PLAYS SPEAKER 2":PRINT
50 PRINT"F3 PLAYS SPEAKER 3":PRINT
60 PRINT"F4 PLAYS SPEAKER 4":PRINT
70 PRINT"F5 LETS YOU ENTER A NEW NOTE
   NUMBER":PRINT
100 PRINT"INPUT A NOTE NUMBER BETWEEN
128 AND 255":INPUTA
120 GETM$:IFM$=""THENGOTO120
130 IFM$=CHR$(133)THENX=S1:GOTO500
140 IFM$=CHR$(137)THENX=S2:GOTO500
150 IFM$=CHR$(134)THENX=S3:GOTO500
160 IFM$=CHR$(138)THENX=S4:GOTO500
170 IFM$=CHR$(135)THENGOTO10
180 GOTO 120
500 FORT"1TO200:POKEX,A:NEXTT:POKEX,0:
   GOTO120
```

Type RUN and hit RETURN.

Now, this little program is just one example of how you can assign different function keys to perform different functions. In this case, we followed some of the general rules we've already established, and introduced a couple of new programming techniques. Let's examine the program, line by line.

LINE 10: contains our "opening message." We also took advantage of the extra space on the line to turn the volume to its highest level (POKE36878,15) and assigned some easy-to-remember variables to each of our four speakers, S1, S2, S3 and S4. Obviously, it's easier to type S1 than 36874, and it also saves memory.

LINE 20 is a blank line.

LINES 30–70 provide instructions for using the program.

LINE 100 is a special INPUT instruction which asks for a NOTE VALUE to be entered into the computer. The INPUT statement assigns the value A to whatever number was typed in by the user.

LINE 120 is our GET statement to check the keyboard.

LINES 130–160 are IF . . . THEN statements which match the messages in lines 30–60. Notice that we could have written EACH LINE like this:
IFM$=CHR$(133)THENFORT=1TO200:
POKES1,A:NEXTT:POKES1,0:GOTO120.
Instead, we wrote a more efficient program which put a "GENERALIZED SOUND ROUTINE" on line 500 so all we have to do in lines 130–160 is define X as the proper speaker and GOTO line 500 to execute the note, then jump back (GOTO) to line 120 to check the keyboard again.

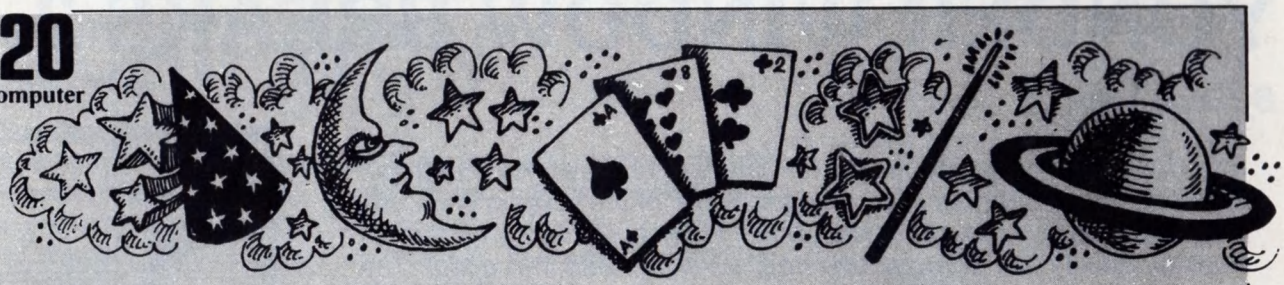
LINE 170 tells the VIC to go back to the beginning of the program and ask for another note value if the user types F5.

LINE 180 tells the VIC to go back and check the keyboard.

LINE 200 contains a time delay loop which specifies how long the note is played when you press the function key (try changing the number 200 to something else). POKE X,A means to POKE the speaker defined as X in lines 130–160 with the note defined as A by the INPUT statement in line 100. In other words, if you entered the number 201 and pressed function key 1, the VIC responds by playing the "D" note from its middle octave. (see your user's manual for note values)

For More Advanced Programmers

There is ANOTHER way to check the keyboard to see which KEY is being held down, using the command **PRINT PEEK (197)** or **PRINT PEEK (203).** You can



PEEK into either of these special memory locations to find out which KEY is being held down. Note that by KEY we mean the physical key being held down, NOT THE SYMBOL. In other words, F1 and F2 are interpreted as being **THE SAME KEY** because we are detecting the physical key is being mechanically held down. The values for the function keys are given below (a complete chart of values for all keys is given on page 179 of the VIC-20 PROGRAMMER'S REFERENCE GUIDE).

VIC KEY	VALUE RETURNED WHEN YOU PEEK (197) or PEEK (203)
no key	64
f1/f2	39
f3/f4	47
f5/f6	55
f7/f7	63

How to use this technique will be covered in a future VIC MAGICIAN, but the general principle involves PEEKing to see if a key is being held down and if the value returned matches the value of a specific key, you make your program perform a specific action. For example, you might write a music program which tells the VIC to play certain notes when certain keys are being held down, and to stop playing those notes when the keys are released. Here's a short program to start you off . . . when you RUN this program, it will play a note when you hit the f1/f2 key. If you get the PEEK values for ALL the keys from the PROGRAMMER'S REFERENCE GUIDE, and match them to the proper note values from the TABLE OF MUSICAL NOTES, you can write your own "VIC PIANO!"

```
10 POKE 36878,15:S1=36874:S2=36875:S3=36876
20 IFPEEK(197)=39THENFORT=1TO50:
   POKES1,200:NEXTT:POKES1,0
30 IFPEEK(197)=47THENFORT=1TO50:
   POKES2,200:NEXTT:POKES1,0
40 IFPEEK(197)=55THENFORT=1TO50:
   POKES3,200:NEXTT:POKES1,0
30 GOTO20
```

The key elements in Line 10 are turning on the volume and defining the "speakers" as S1, S2 and S3. In lines 20–40 we PEEK (197) to see what key is being held down, then insert a time delay loop (1to50) to specify how long each note is held down, then POKE a note value into the speaker we want, then we turn OFF the speaker (otherwise it would keep playing). Change the number 50 to a higher or lower number to increase or decrease the duration each note is played.

Summary

This introduction to the VIC's programmable function keys is only a beginning. You can probably design programs which do much more than those described here . . . for example, you could make the function keys stand for different colors, and instead of POKEing note values, designate special keys to POKE different COLOR COMBINATIONS using the SCREEN AND BORDER COLOR chart in your user's guide.

You could specify each function key to perform a different complex calculation by combining the function keys with DEF FN statements. You might want to INPUT a series of numbers and perform several multi-step calculations which you can execute simply by pressing the function keys.

You might match the function keys to program SUBROUTINES which allow the user to access different portions of a long complex program, so he can jump from a main program into a smaller subroutine, and then come back to the main program . . . all by pressing one or two function keys.

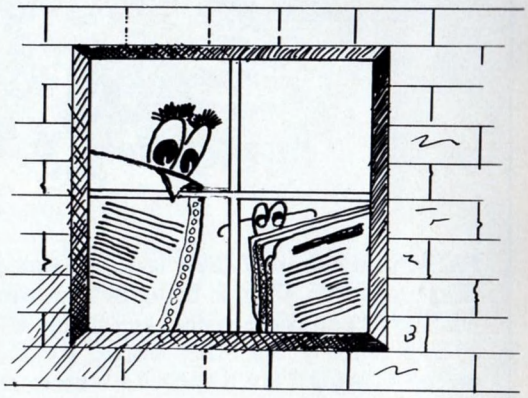
Finally . . . you may have already discovered that programmability isn't limited to the VIC's function keys. In fact, ALL the VIC's alphanumeric and graphic keys are programmable, using the techniques described here in combination with the CHR$ values for each key. To program a key to perform a special function, simply use that key's CHR$ value just like the function key values described here.

If you develop an imaginative application for the VIC's special function keys, drop a letter to THE VIC MAGICIAN in care of our magazine and we'll share your discoveries with the rest of the "VIC world." **Enjoy!** C=

Two COMMODORE cartridge products which offer built-in function key programming include the SUPEREXPANDER cartridge and the PROGRAMMER'S AID CARTRIDGE. Both cartridges let you redefine the preprogrammed function keys by typing "KEY" and hitting RETURN. See your Commodore Dealer for more information.

HARK WHAT TEXT THROUGH YONDER WINDOW BREAKS ?

By Dirk Williams.



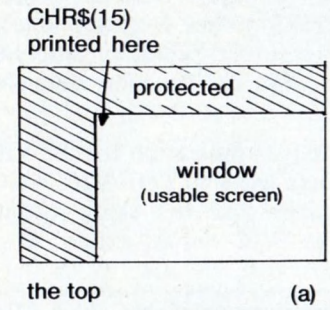
The CBM 8032 certainly has it's fair share of features. Among these is the facility to define the 'scrolling window' of the screen. What this means is that you can set aside a box on the screen in which normal screen activity (i.e. printing, clearing, scrolling, etc) can occur. The rest of the screen (that which is outside the window) remains immune to normal scrolling, clearing activity. You could for example, have a permanent border or simply 'protect' a section of the screen from alteration. When you print, the characters will appear in the box – and will scroll off the top of the box (not into the protected screen) if scrolling occurs. A clear screen command will clear only the window, thus leaving the 'protected' portion unchanged.

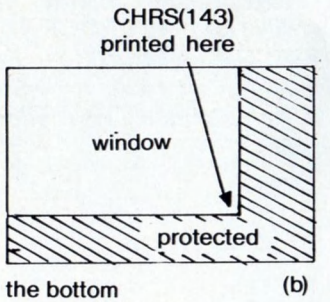
There are two control characters that allow us to allocate this window - one for the top and lefthand borders and another for the bottom and righthand borders. The borders are set when the appropriate characters are printed in the desired locations on the screen. Note, however, that the borders can **not** be seen.

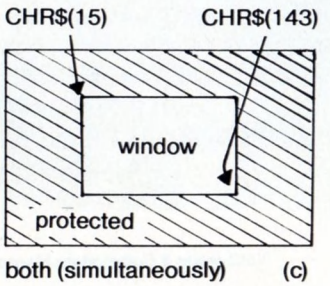
top and left borders – CHR$(15)
bottom and right borders – CHR$(143)

how the borders work:

Fig. 1





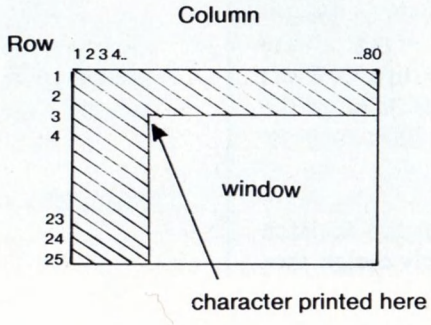


The border should be set from within a Program (although borders still operate in immediate mode once set). They may be cancelled from either immediate mode or from within a program by simply pressing HOME two times, then CLR screen – or by printing these control codes (i.e. PRINT "[HOME key 2 times] [SHIFT CLR/HOME key]". You may cancel, set, move the size of the screen window at any time from within a program – but you must cancel any existing windows first. You don't have to set both borders, in fact you could create a window such as that in FIG. 1(a) and (b) – by only setting one of the characters.

In order to set the borders, you must position the cursor 'above' the screen location you wish to constitute one of the window corners (as shown in FIGURES 1). For example, if you wished to define a window 4 columns across and two rows from the sides of the actual screen. Remember that anything you have outside the window when it is set, will be 'preserved' – so get rid of any garbage first.

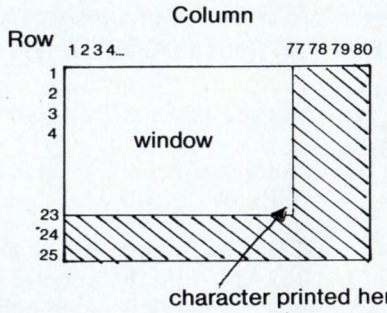
SETTING THE TOP

```
10 PRINT"[home][2 csr dn][4 csr right]";CHR$(15)
```



SETTING THE BOTTOM

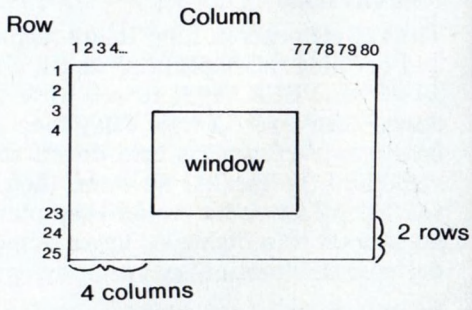
```
20 PRINT"[home]";:FORX 1to22:PRINT"[csr dn]";:NEXT
30 FORX=1to7:PRINT"[csr right]";:NEXT:PRINTCHR$(193)
```



This is a small trick when combining the two borders to create a 'suspended' box. When the top border is set (this should be done last) – the home command sends the cursor to the corner of the box, not the screen. Thus if we were to set this first, we would have to take into account how far down the top border is when moving the cursor down to set the bottom. Thus if we set the bottom first, our cursor can be returned to the normal home position and moved down as normal to set the top.

COMBINED BORDERS (WINDOW)

```
10 PRINT"[home][2 csr dn][4 csr right]";CHR$(15)
20 PRINT"[home]";:FORX=1to22:PRINT"[csr dn]";:NEXT
30 FORX=1to7:PRINT"[csr right]";:NEXT:PRINTCHR$(193)
```



Try one of the above programs and when you return to immediate mode, move the cursor around and see what effect the window has.

WATERLOO MICRO SOFTWARE DESCRIPTION

An extensive software package has been developed to satisfy many of the educational requirements at the University of Waterloo. This portable software is particularly suited to microcomputers, but identical versions will be available on medium and large-scale computing facilities. Thus a user is not limited by the capacity of the micro; the identical progam will run without modification on larger and faster equipment.

The package consists of interpreters for various languages, an editor, an operating system (supervisor) and an assembly language development system.

a) Language interpreters are included for four (4) programming language dialects known as:

Waterloo microBASIC
Waterloo microPASCAL
Waterloo microFORTRAN
and Waterloo microAPL

These language interpreters have been designed specifically for educational use in the teaching of computer programming. The design of the interpreters features good error diagnosis and debugging capabilities which are useful in educational and other program-development environments.

Waterloo microBASIC includes ANS Minimal BASIC, with a minor exception, and several extensions such as structured progamming control, long names for variables and other program entities, character-string manipulation, callable procedures and multi-line functions, sequential and relative file capabilities, integer artihmetic, debugging facilities, and convenient program entry and editing facilities.

Waterloo microPASCAL is an extensive implementation of Pascal, corresponding very closely to draft proposals being produced by the International Standards Organisation (ISO) Pascal committee. The ISO draft language is a refinement of the language defined by Wirth, varying only in minor aspects. This implementation includes sophisticated features such as text file support, pointer variables, and multi-dimensional arrays. A significant feature of Waterloo microPASCAL is its powerful interactive debugging facility.

Waterloo microFORTRAN is a special dialect designed for teaching purposes. It has many of the characteristics and much of the flavour of normal FORTRAN, but varies significantly from established standards for that language. This language processor has many of the important characteristics of the WATFIV-S compiler which is widely used on IBM computers, plus some features from the new FORTRAN-77 definition. Examples of language features supported are FORMAT, subroutines and functions, multi-dimensional arrays, extended character-string manipulation, structured programming control and file input/output. In addition, the interpreter provides a powerful interactive debugging facility.

Waterloo MicroAPL is intended to be a complete and faithful implementation of IBM/ACM standard for APL with respect to the syntax and semantics of APL statements, operators and primitive functions, input and output forms, and defined functions. System commands, system variables and system functions are those consistent with a single user environment. There are no significant design limitations on the rank or shape of arrays or the length of names. The shared varible processor is omitted. Extensions include system functions supporting files of APL arrays. APL equivalents of the BASIC features PEEK, POKE and SYS are included.

b) A text editor known as Waterloo microEDITOR, is suitable for creating and maintaining both program source and data files. It is a traditional line-orientated text editor with powerful text searching and substitution commands including global change. Full-screen support and special function keys allow text to be altered, inserted and deleted on the screen without entering commands. Facilities for repeating and editing previously issued commands further enhance

the useability of this editor.

c) File-orientated Assembler and Linker programs, known as the Waterloo 6809 Assembler and Linker, are included which support development of general-purpose Motorola 6809 machine-language programs. The Assembler supports syntax and directives for Motorola 6809 assembly language and includes powerful macro capabilities. In addition, the Assembler supports pseudo opcodes for structured programming control, long names (labels) for meaningful identification of program segments and data, and the ability to include definitions from separate files. The Assembler produces both a listing and a relocatable object file.

The linker allows the combination of an arbitrary number of relocatable object files to produce an absolute

continued on page 35

BASIC Plotter

by
Paul Higginbottom

This program will plot random lines using the "quarter-square" graphics characters. Although it's a program in itself, it could easily be made into a subroutine.

The program has been set up for 80 column screens (line 9040). Notice "LL" (Line Length) is multiplied by 2 in lines 2020 & 2030? Since the quarter squares use up half a character space in the "x" direction, an 80 column screen can have up to 160 "half-characters" horizontally. Similarly, on 25 lines there can be up to 50 half characters vertically ("y" direction). For 40 column screens you'll need to change LL to 40; the second parameter remains the same since both have 25 lines.

Line 2000 clears the window (if one set), the screen, and sets graphics mode (no gap between lines). If you like, substitute CHR$(142) with 'esc-rvs-N' and stick it inside the quotes.

```
2000 PRINT"[HM HM CLR]"CHR$(142)
2010 GOSUB 9000
2020 X1=INT(RND(TI)*LL*2) :
   Y1=INT(RND(TI)*50)
2030 X2=INT(RND(TI)*LL*2) :
   Y2=INT(RND(TI)*50)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : GOTO 2030
3000 REM ********** PLOT A LINE **********
3010 DX=X2-X1 : DY=Y2-Y1 : X=X1 : Y=Y1
3020  L=SQR(DX*DX+DY*DY)  :  IF  L=0  THEN
      3040
3030 XI=DX/L : YI=DY/L
3040 GOSUB 8000 : IF (ABS(X2-X)<=ABS(XI))
   AND (ABS(Y2-Y)<=ABS(YI)) THEN RETURN
3050 X=X+XI : Y=Y+YI : GOTO 3040
8000 REM ********** PLOT X, Y **********
8010 TX=INT(X+IR):TY=INT(Y+IR)
   :SQ=AM(TX AND AM, TY AND AM)
8020 P=BS+TX/DV-INT(TY/DV)*LL : POKE P,
   C(I(PEEK(P))OR SQ) : RETURN
9000 REM ********** SETUP **********
9010 DIM C(15), I(255), AM(1,1)
9020 FOR I=0 TO 15 : READ C(I) : I(C(I))=I : NEXT
9030 FOR I=0TO1 : FOR J=0TO1 :
   AM(J,I)=(J+1)*4↑I : NEXTJ,I
9040 LL=80 : BS=32768+24*LL : DV=2 : AM=1 :
   IR=.5
9050 DATA 32, 123, 108, 98, 126, 97, 127, 252, 124,
   255, 225, 254, 226, 236, 251, 160
9060 RETURN
```

The subroutine at 9000 sets up an array with the 16 possible combinations of the quarter squares. BS is the base address or the POKE address of the bottom left corner of the screen.

All plotting efforts are performed by the two subroutines at 3000 & 8000. Subroutine 3000 plots a line from x1,y1 to x2,y2 by plotting several points (sub 8000). At the same time, subroutine 8000 must determine if there is already a point in a character space. If there is, the POKE information must not interfere with existing points. Lines 200X are used for plot criteria generation. The above merely plots random lines. For something more meaningful, try substituting with these:

```
2020 X1=0 : Y1=1
2025 FOR X2=0 TO 159
2030 Y2=EXP (X2/31.4)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : NEXT : END
```

```
2020 N=6 : C=3.1415926/160 : X1=0 : Y1=25
2025 FOR X2=0 TO 159
2030 Y2=25 + 24 * SIN(X2 * N * C)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : NEXT : END
```

```
2020 N=8 : C=3.1415926/160 : X1=0 : Y1=50 :
   DC=100
2025 FOR X2=0 TO 159
2030 Y2=25 + 24 * COS(X2 * N * C) * EXP(-X/DC)
2040 GOSUB 3000 : Y1=Y2 : X1=X2 : NEXT : END
```

The first plots an exponential curve. Notice the Y origin is set to 1 rather than 0. This accounts for a slight inaccuracy as the plotter draws horizontal lines using the top "half-character" rather than the bottom half-character. This could be changed by modifying the character table at 9050.

The second draws a SINE curve starting half way up the screen (Y1=25). The variable N represents the number of half cycles displayed (N=6 will draw 3 complete cycles).

The last one is a decaying COSINE wave, origin at top-left (Y1=50). For higher decay rates, use lower values in DC.

Finally, with little effort you could use the plotter routine to draw axes for your functions. C=

SuperPET

by Richard Black,
PO BOX 702
CANBERRA 2601
ph (062)49-4692

The SuperPET 9000 is an extension of Commodore's 8032 computer to include a second processor (Motorola 6809) which runs (pseudo-) 16-bit code. Provided for the 6809 is the Waterloo Software Package which includes 6809 Assembler, good editing (and debugging) facilities and higher level languages such as BASIC, FORTRAN, Pascal, and APL. An overview of this hybrid 6502/6809 system is given. We make:-

1) general comments about the Waterloo Software (especially with regard to Pascal), and
2) suggestions for future developments/improvements.

1. INTRODUCTION

A short time ago, when considering using a first generation microcomputer, a worthy question for those who had ready access to larger facilities was:- 'But, what can you do with it?' – Building one's own system could have been interesting... Certainly micros were good for:-

0. game playing;
1. providing a cheap introduction to computers for those who would not otherwise have ready access;
2. providing ready made packages (e.g. for businessmen, etc) such as wordprocessing, VISICALC, small data base management systems (e.g. OZZ and THE MANAGER);
3. for interfacing to experiments and as controllers, recorders (especially the PET with it's IEEE-488 bus).

Now, the 16-bit code provided by Waterloo provides excellent editing/language facilities. Editing is numeric keypad style- similar to (although not quite as many features as) that on Digital's VT-100 terminals. Languages include versions of Assembler, BASIC, FORTRAN, Pascal, and APL (with COBOL and Ada in the pipeline) designed for structured programming. We have now reached a transition phase whereby the software development activities are at least as good as those that we would expect on many larger machines.

Impressions in this report have been gained from a SUPERPET being used in farming applications in New Zealand.

HARDWARE AND CONSEQUENCES

The innovation Commodore have made with the SUPERPET is the inlcusion of a pseudo 16-bit MC6809 based system. The Motorola 6809 has been described as "the most powerful of 8-bit processors" (really an 8/16 bit hybrid -internally it is 16-bit), and can be regarded as the choice for those who want to get into structured programming. In the hardware line other manufacturers (e.g. Hitachi, Tandy) are following suit in adopting the 6809-bit processor, but this is a far cry from a software package that has undergone nearly three years of extensive university testing and development.

Now we have:-

a) (Near) compatibility with software on larger machines (which to a large degree is 16-bit code) e.g. the IBM 370 and PDP-11/45 series.
b) With the inclusion of the Commodore 8032 system, we retain compatibility with software already developed.

Certainly the 6809 solution seems a good stepping stone – one wonders when they will introduce a true 16-bit processor (Gowans, 1981; Toong and Gupta, 1981), for example:-

a) Motorola 68000 (which has an addressing space of 64M cf.

64K for 8-bit machines without bank-switching, and can be thought of as having the power of a PDP-11/45 on a single chip),

b) their own – at one stage it was romoured (Gowans, 1981) that '... Commodore chip production subsidiaries MOS Technology and Frontier Semiconductor will bring out a machine of their own that might – just might – run 6502 code as well as 16-bit...' Note that although 16-bit processors have superior memory addressing capabilities, speed and instruction sets, one has to have the software and support chips to take advantage of these. For example, benchmarks (although these need to be interpreted with caution) do not show a significant increase in speed (cf. 8-bit machines) of the BASIC for the IBM Personal Computer (the 16-bit processor is Intel's 8088 – a series introduced two years before the 68000 with a direct memory addressing capability of 1M (not that IBM make full use of this) – really an improved version of the 8-bit 8080-although note that Intel with its 432 - programmed in Ada - already has 32-bit processors.

3. EDITING

3.1 6502:CBM BASIC

Editing/debugging can be significantly enhanced by the POWER programmers utility package (Professional Software Ltd.)

3.2 6809:Waterloo microEDITOR

Program development is considerably facilitated by 'keypad-style' editing (cf. Digital's VT-100 terminals). Arrow keys allow the positioning of the (screen) cursor anywhere in the file being edited. All essential features for fast editing are then obtained by pushing e.g. a single ('shifted') key on the numeric keypad – - e.g.

– insert/delete line(s), character(s) – command/screen mode.

In command mode one can e.g. change (c/string1/string2/) or search (string) selected/all occurences of strings; etc. Put/get (p/g) allow disk files to be created and recovered. Blocks of text can be moved from one place to another by e.g. '10,15 p temp' (puts lines 10 to 15 into a disk file called 'temp'); 'g temp' (gets the lines back again when the sreen cursor has been positioned in the place that they are required).

4. LANGUAGES

4.1 6502

As well as the Commodore BASIC 4.0 supplied in ROM one can obtain:-

1. COMAL – simple structured language;
2. PETSPEED BASIC compiler (from Oxford Computer Systems (Software) – who claim that compiled programs run 'at up to 30 times the speed of their interpreted equivalent in PET BASIC; and who also supply COMPILED INTEGER BASIC which runs at 'up to 150 times the speed of PET BASIC');
3. DTL BASIC Compiler
4. TCL Pascal (compiled);
5. USCD Pacsal
6. FORTRAN
7. LISP (See Commodore Encyclopedia of Software).

If one decides to go in for the 8-bit disk operating system CP/M via Softbox (Small Systems Engineering Ltd)., then a large range is available including ALGOL, C, muLISP, COBOL, PL/1 and the symbolic/semi-numeric processors muSIMP and muMATH for computer algebra).

4.2 6809

With the aim of providing '... good primitives for structured programming...', Waterloo Computing Systems (at the University of Waterloo) have produced a software package which includes (along with good editing and debugging facilities) 6809 Assembler and interpreters for the higher level languages BASIC, FORTRAN, Pascal and APL, with COBOL AND Ada to come.

4.2.1 Waterloo microBASIC

This is a superset of the usual BASIC which is designed for structured programming and includes:- indentation, long identifiers, procedures (e.g. 'FindChangOfSign' is more meaningful than 'GOSUB' 900'). Control words include e.g. endo, endif, quitif, etc. Waterloo are to be congratulated on the job that they have done here, and it is to be hoped that all microcomputers will have a language similar to this or COMAL (as a minimum) in ROM.

4.2.2 Waterloo microFORTRAN

This is a variant of FORTRAN-77 '...designed for educational and research environments...'

4.2.3 Waterloo microAPL

This is a complete implementation of the IBM/ACM standard for APL.

4.2.4 Waterloo microPascal

This implementation of Pascal is fairly close to the proposed ISO standard, although it does omit two features, which I hope will be included in future versions:-

a) Passing of functions and procedures as parameters – e.g. In numerical analysis one commonly wants to carry out the same procedure on different functions – for example:-

```
program passing-functions-as parameters;
var
   zero-of-g, zero-of-k: real;
procedure newton( function f( x: real ): real; xtrial: real; var xzero: real );
   begin
   < find xzero such that f(xzero)=0 >
   end < newton gt
.
.
< Declaration of functions g and k go here >
.
.
begin
   <Find zeroes of g and K>
   newton( g,3.8, zero-of-g );
   newton(k, 2.1, zero-of-k )
end.
```

FIGURE 1

(Note that the above declaration (Fig. 1) of f (Addyman, 1981) required in the ISO proposal, avoids the passing of functions with the incorrect

number or type of parameters; although this could be achieved by 'function f(real): real'; as in DEC-10 Pascal. The original Jensen and Wirth definition merely required 'function f: real', which could have lead to run-time rather than syntax errors. Maybe the previous lack of standardisation led Waterloo to delay implementation of this feature for the sake of portability.)

b) Upper and lower case identifiers:

The one obvious difference between the Waterloo microEditor and the editor for microPascal is that the latter converts all upper case letters typed into the original file to lower case (except for those enclosed by inverted commas e.g. 'A'). The manual mentions that the interpreter treats cases as equivalent i.e. A=a, and indeed readability, the above-mentioned conversation is unnecessary. As a mitigating factor, it should be mentioned that Waterloo do allow the underscore character as a word separator within identifiers – i.e. file-of-integer is allowed, whereas FileOfInteger gets converted to fileofinteger.

In summary, it would be nice to be able to say, e.g. procedure FindChangeOfSign(function f(x:real):real;

```
        xmin, xmax:real;
      var xChangeMin,
          xChangeMax: real );
```

Apart from the above all other standard features are implemented (except for two minor details – pac and unpack, ...). Extensions are the following:-

a) Peek and poke are allowed.

b) As well as having the standard forms of 'reset' and 'rewrite', these have been extended to include e.g.

reset(disk-file-name, pascal-file)

where file-name is of type 'packed array [1..16] of char' (and may also include the disk drive and device number, if the default is not used).

To testify the ease of software developement using the editor and Pascal debugger – during my first weekend using the machine, programs developed included a text-processor, and music-player. I proceed to comment on these.

PROGRAMMING EXAMPLES

A) Wordprocessing:

As a start one could use the microEDITOR – this is a big leap forward from the typewriter, and sufficient for many – i.e. one can enter text as fast as one can type, and manipulate it in the ways described in section 3. But, if one wants to add the various 'trendy' features such as automatic justification then:-

i) There are available a number of good packages (written in machine-code, and thus relatively fast), or

ii) One can write one's own – a very good programming exercise exemplifying character manipulation. Ideally one would like to be able to write a program in a higher level language (with clarity being paramount), yet have it run reasonably fast. The microPascal program developed (an extension of that described in ref. 3) converted a file (entered using the microEditor) into a word-processed (justified, filled, paragraphed etc.) at a rate of 2-3 lines per minute – for my purposes this was quite satisfactory in that if left overnight it would produce over 30 pages of formatted text.

It is appropriate to mention here that TCL Pascal for the 6502 does have a number of points in its favour (although the microEditor is not available for it)

i) Passing of functions/procedures as parameters, and mixed upper/lower case identifiers are allowed.

ii) It is compiled rather than interpreted – thus, after the initial compilation, one has an object-code program which will run faster. For example, if the above text-processor ran ten times faster, then a speed of one page per minute would be quite respectable.

B) MUSIC: – a frivolous example:

The 4000/8000/9000 series contain a small speaker which can be programmed for various sound effects.

My eleven-year old sister has gained much amusement from a Pascal program in computer-music composition. Ideally one would like to define a function 'time' which peeks the appropriate memory location to find the time (e.g. in jiffies) thus providing a more accurate timer (for length of notes) than my crude while loop.

5. CONCLUSIONS

When considering what computing facilities to use, ease of software development is paramount, and the SUPERPET scores better than many mainframe systems. For smaller jobs this system should satisfy many of the needs of those in educational and research environments (especially in the teaching of undergraduate Computer Science). For larger jobs the SUPERPET can be used as an intelligent terminal for some host system.

On the hardware side, we have the convenience and economic sense of the concept of more than one processor – compatibility with 6502 software is retained, while the 6809 provides a good stepping stone to the software of 16-bit mainframes, and 16-bit processors such as the 68000 (which contains the CPU power of a PDP-11/45 on a single chip).

In this dynamic world of microprocessing, one can always wait for bigger and better systems but there is a transition stage when the average person can really start doing things, and that occurs here with a (pseudo) 16-bit based system.

for references see page 35

USING THE COMMODORE PRINTER

By Jim Butterfield.

When you are producing output, it's good to make it neat. The computer is there to help its human readers, and the more you can do to improve the information, the better job you'll be doing.

Printing in Columns

Beginners often arrange values in columns by using the screen tabulation functions: putting a comma into the PRINT statement, or using the TAB function. These methods work, but they have a pitfall: they won't behave properly if the output goes to other devices. The problem is that the computer always knows exactly where the screen cursor is, but it never knows on what column the external devices are located. It doesn't even try to keep track; so a TAB or a comma directed to the printer or other device won't behave properly.

It's my feeling that almost everything that goes the screen can be usefully directed to the printer, or written to a disk file with a view to transferring to the printer later. Once you have a report looking nice on the screen, you don't want to reprogram to get it looking nice in print. So... stay away from TAB and commas – there's a better way.

Redirecting Output

While I'm on the subject of switching output from the screen to the printer, I'd like to share a little coding trick with you. Most programmers know that you can direct output to a printer by performing an OPEN to device number 4 (the printer) and then using PRINT#... That's fine for a finished program, but you can waste a lot of paper while you're checking out a program if you do everything to the printer.

Here's a trick: We can OPEN to device number 3 (the screen) and PRINT# to the screen, checking our program and fixing it up. When it's ready to go, all we need to do is to change the OPEN statement so that it names device number 4, and output goes to the printer. We save time and paper. Let's try it: we code:

```
100  OPEN 1,3
110  FOR J=1 TO 10
120  PRINT#1,J;SQR(J)
130  NEXT J
140  CLOSE 1
```

When we run this program, output is delivered to the screen. If everything looks good, we can change line 100 to OPEN 1,4 ...and output is redirected.

It's not really a trick; it's good coding. We could allow the user to specify what output he wanted by coding something like:

100 INPUT"DEVICE NUMBER";N :OPEN 1,N so that the user could type in 3 or 4 to select the type of output he wants.

Neatness Counts

If I'm sternly discouraging TAB and comma, how can you arrange things in columns? A few simple answers, but first some ground rules. The best way to arrange stuff in columns is to make sure that each "field" is always the same length; that way, each item will be printed neatly in the same place across the page.

How can we rechop two numbers as different as 3 and -32768 so that they occupy the same space? For that matter, how can we take two names as BUTTERFIELD and NG and make them the same length?

Let's take the names first. These "strings" could be neatly chopped down to a fixed length by means of the LEFT$(function ...if they were long enough. For example, we could slice out the first eight characters of string X$ with LEFT$(X$,8); but it won't work if X$ is less than eight characters long in the first place. So – pay attention – we must first pad out the name by adding spaces to the end. Sticking extra characters onto the end of a string is called "concatenation" – pronounced with emphasis on the cat – and is done with a plus sign. If we had a short name like M and wanted to tack eight spaces on the end, we'd do it by writing "M"+" " which would create a new string nine characters long. A name like BUTTERFIELD treated the same way would end up nineteen characters long, but this doesn't matter: we're going to chop them both down to the same length with LEFT$(.

Let's put it all together. If the name is held in variable N$, we code PRINT LEFT$(N$+" ",8); with a semicolon at the end. First we concatenate, adding the spaces; then we chop (or "truncate"), cutting to a fixed length; finally we print. Both long and short names will be printed as exactly eight characters; the next thing we print will be neatly lined up behind it. We might want to make the field more than eight characters long, since a splendid name like BUTTERFIELD would end up chopped to BUTTERFI – if we do increase the length we must remember to add more spaces, of course.

The above procedure is called Left Justification, since the strings are lined up neatly on the left with spaces filling out on the right hand side. We can go the other way and produce Right Justification with a small adjustment: try PRINT RIGHT$(" " +N$,6); and you'll see how the left side fills with spaces and names line up on the right. This is the kind of alignment you will want with numbers; we'll deal with that in a moment. Remember that if you don't allow enough space you'll end up with chopped-off names like TERFIELD, and there's no justification for that...

If the numbers you are using are integers, you'll usually want to line them up with right justification. Once again, this is easy to do once you know the function that changes numbers to strings. If your value is held in variable X, we can change it to a string with STR$(X); now we can do the right justification with PRINT RIGHT$(" " +STR$(X),6); everything will work out neatly. Study this statement and see how X builds up into a neatly furnished string of length six.

If your numbers contain fractional values, you may want to try to line up the decimal points. That's much more challenging. Perhaps you'd like to try your hand at it. We'll tackle it here another time.

MicroPro Design Pty.Ltd.

Specialising in the sales & support of the Commodore PET/CBM Microcomputers, peripherals and interfaces, including:

- IEEE488-RS232 COMMUNICATIONS
- RS232/CENTRONICS PRINTER INTERFACE
- EPROM PROGRAMMER
- WORD PROCESSOR PRINTER INTERFACE

For full details and prices call or write:

205/6 Clarke St
CROWS NEST
Ph:(02) 438 1220

Postal: P.O. Box 153
NORTH SYDNEY
NSW 2060

PHOTOTYPESETTING

★ Phototypesetting from text disks and/or cassettes generated from Commodore microcomputers.
★ Complete art studio facilities
★ Reports, software manuals, advertising, etc. etc.

(02) 439 1827

MERVYN BEAMISH GRAPHICS
82 Alexander St. CROWS NEST, NSW 2065

VIC-20

40/80 COLUMN CARTRIDGE

★ 40/80 Column Display (not colour)
★ Vic and CBM Graphics
★ Load all PET/CBM Programmes
★ All Cursor control Upper and Lower Case
★ Switchable from basic without losing programme
★ No alterations to Vic required
★ No external power supply required
★ Works together with 32k Basic Vic computer

ELECTRONICS

416 LOGAN RD. (Pacific Hwy.) STONES CORNER,
BRISBANE. TEL: (07) 397 0808, 397 0888. P.O. Box 274
SUNNYBANK QLD. 4109 TELEX AA40811

THE VIC CENTRE

BASIC & OPERATING SYSTEM VARIABLES for VIC

NAME:	FUNCTION:
C000-C045	Action addresses for primary keywords
C046-C073	Action addresses for functions
C074-C091	Hierarchy and action addresses for operators
C092-C192	Table of BASIC keywords
C193-C2A9	BASIC messages, mostly error messages
C38A-C3B7	Search stack for FOR or GOSUB activity
C3B8-C3FA	Open up space in memory
C3FB-C407	Test: stack too deep?
C408-C434	Check available memory
C435	Send canned error message, then:
C474-C482	Print Ready
C483-C532	Handle new BASIC line from keyboard
C533-C55F	Rebuild chaining of BASIC lines in memory
C560-C57B	Receive line from keyboard
C57C-C612	Change keywords to BASIC tokens
C613-C641	Search BASIC for given BASIC line number
C642	Perform NEW, then:
C660-C68D	Perform CLR
C68E-C69B	Reset BASIC execution to start-of-program
C69C-C741	Perform LIST
C742-C7EC	Perform FOR
C7ED-C81G	Execute BASIC statement
C81D-C82B	Perform Restore
CB2C-C856	Perform STOP and END
C857-C870	Perform CONT
C871-C882	Perform RUN
C883-C89F	Perform GOSUB
C8A0-C8D1	Perform GOTO
C8D2-C8EA	Perform RETURN, and perhaps:
C8EB-C905	Perform DATA, i.e., skip rest of statement
C906-C908	Scan for next BASIC statement
C909-C927	Scan for next BASIC line
C928-C93A	Perform IF, and perhaps:
C93B-C94A	Perform REM, i.e., skip rest of line
C94B-C96A	Perform ON
C96B-C94A	Get fixed-point number from BASIC
C9A5-CA1C	Perform LET
CA1D-CA2B	Add ASCII digit to accumulator No. 1
CA2C-CA7F	Continue to perform LET
CA80-CA85	Perform PRINT #
CA86-CA99	Perform CMD
CA9A-CB1D	Perform Print
CB1E-CB3A	Print string from memory
CB3B-CB4C	Print single format character (space, cursor-right?)
CB4D-CB7A	Handle bad input data
CB7B-CBA4	Perform GET
CBA5-CBBE	Perform INPUT No.
CBBF-CBF8	Perform INPUT
CBF9-CC05	Prompt and receive input
CC06-CCFB	Perform READ; common routines used by INPUT and GET
CCFC-CD1D	Messages: EXTRA IGNORED, REDO FORM START
CD1E-CD77	Perform NEXT
CD78-CD9D	Check data type, print TYPE MISMATCH
CD9E-CEF0	Input & evaluate any expression (numeric or string)
CEF1-CEF6	Evaluate expession within parentheses()
CEF7-CEF9	Check right parenthesis)
CEFA-CEFC	Check left parenthesis(
CEFD-CF07	Check for comma
CF08-CF0C	Print SYNTAX ERROR and exit
CF0D-CF13	Set up function for future evaluation
CF14-CFA6	search for variable name

NAME:	FUNCTION:
CFA7-CFE5	Identity and set up function references
CFE6-CFE8	Perform OR
CFE9-D015	Perform AND
D016-D07D	Perform comparisons, string or numeric
D07E-D08A	Perform DIM
D08E-D112	Search for variable location in memory
D113-D11C	Check if ASCII character is alphabetic
D11D-D193	Create new BASIC variable
D194-D1A4	Array pointer subroutine
D1A5-D1A9	32768 in floating binary
D1AA-D1D0	Evaluate expression for positive integer
D1D1-D34B	Find or create array
D34C-D37C	Computer array subscript size
D37D-D390	Perform FRE then:
D391-D39D	Convert fixed point to floating point
D39E-D3A5	Perform POS
D3A6-D3B2	Check if direct command, print ILLEGAL DIRECT
D3B3-D3E0	Perform DEF
D3E1-D3F3	Check FNx syntax
D3F4-D464	Evaluate FNx
D465-D474	perform STR$
D475-D486	Calculate string vector
D487-D4F3	Svan and set up string
D4F4-D525	Subroutine to build string vector
D526-D5BC	Garbage collection subroutine
D5BD-D605	Check for most eligible string collection
D606-D63C	Collect a string
D63D-D679	Perform a string concatenation
D67A-D6A2	Build string into memory
D6A3-D6DA	Discard unwanted string
D6DB-D6EB	Clean the descriptor stack
D6EC-D6FF	Perform CHR$
D700-D72B	Perform LEFT$
D72C-D72C	Perform RIGHT$
D737-F760	Perform MID$
D761-D77B	Pull string function parameters from stack
D77C-D781	Perform LEN
D782-D78A	Move from string-mode to numeric-mode
D78B-D79A	Perform ASG
D79B-D7AC	Input byte parameter
D7AD-D7EA	Perform VAL
D7EB-D7F6	Get two parameters for POKE or WAIT
D7F7-D80C	Convert floating point ot fixed point
D80D-D823	Perform PEEK
D824-D82C	Perform POKE
D82D-D848	Perform WAIT
D849-D84F	Add 0.5 to accumulator No. 1
D850-D861	Perform subtraction
D861-D946	Perform addition
D947-D97D	Complement accumulator No. 1
D97E-D982	Print OVERFLOW and exit
D983-D9BB	Multiply-a-byte subroutine
D9BC-D9E9	Function constants: 1, SQR(.5), SQR(2),-0.5. etc
D9EA-DA2F	Perform LOG
DA30-DA58	Perform multiplication
DA59-DA8B	Multiply-a-bit subroutine
DA8C-DAB6	Load accumulator No. 2 from memory
DAB7-DAD3	Test and adjust accumulators No. 1 and No. 2
DAD4-DAE1	Handle over and underflow
DAE2-DAF8	Multiply by 10
DAF9-DAFD	10 in floating binary
DAFE-DB06	Divide by 10
DB07-DB11	Perform divide-into
DB12-DBA1	Perform divide-by
DBA2-DBC6	Load accumulator No. 1 from memory
DBC7-DBFB	Store accumulator No. 1 into memory
DBFC-DC0B	Copy accumulator No. 2 into accumulator No. 1

NAME:	FUNCTION:
DC0C-DC1A	Copy accumulator No. 1 into accumulator No. 2
DC1B-DC2A	Round off accumulator No. 1
DC2B-DC38	Computer SGN value of accumulator No. 1
DC39-DC57	Perform SGN
DC58-DC5A	Perform ABS
DC5B-DC9A	Compare accumulator No. 1 to memory
DC9B-DCCB	Convert floating-point to fixed-point
DCCC-DCF2	Perform INT
DCF3-DD7D	Convert string to floating-point
DD7E-DDB2	Get new ASCII digit
DDB3-DDC1	String conversion constants: 99999999,999999999 IE+9
DDC2	Print IN, followed by:
DDCD-DDDC	Print BASIC line number
DDDD-DF10	Convert number or TI$ to ASCII
DF11 DF70	Constants for numeric conversion
DF71-DF77	Perform SQR
DF78-DFB3	Perform power function
DFB4-DFBE	Perform negation
DFBF-DFEC	Constants for string evaluation
DFED-E03F	Perform EXP
E040-E089	Function series evaluations subroutines
E08A-E093	Manipulation constants for RND
E094-E0F5	Perform RND
E0F6-E260	Kernal patch routines (See Appendix 6 for listings)
E261-E267	Perform COS
E268-E2BO	Perform SIN
E2B1-E2DC	Perform TAN
E2DD-E30A	Constants for trig evaluation pi/2,2No.pi,.25, etc
E30B-E33A	Perform ATN
E33B-E377	Constants for ATN series evaluation
E378-E386	Initialise RAM vectors
E387-E3A3	Subroutine to be moved to zero page ($70 to $87)
E3A4-E428	Initialise BASIC system
E429-E44E	Messages: BYTES FREE,****CBM BASIC V2****
E44F-E47B	Vector initialisation
E47C-E4FF	Unused space
	KERNAL ROUTINES
E500-E504	Return address of 6522
E505-E509	Return max rows and columns of screen
E50A-E517	Read/plot cursor position
E518-E580	Initialise I/O
E581-E586	Home function
E587-E5B4	Move cursor to current line index pointer
E5B5-E5C2	Panic NMI entry (Restore key)
E5C3-E5CE	Initialise 6560 VIC chip
E5CF-E64E	Remove character from queue
E64F-E741	Input a line until carriage return
E742-E8E7	Print routine
E8E8-E8F9	Check for decrement in line index pointer
E8FA-E911	Check for increment in line index pointer
E912-E928	Check colour
E929-E974	Table to convert from screen code to KASCII
E975-EAA0	Screen scroll routines
EAA1-EB1D	IRQ routines, put char o screen and update time, generate I/O
EBIE-EC45	General keyboard scan
EC46-EE13	Keyboard matrix tables
EE14-EEBF	Command serial bus device to listen
EECO-EEC4	Send secondary address after listen
EEC5-EECD	Release attention after listen
EECE-EEE3	Talk on second address
EEE4-EEF5	Buffered output to serial bus
EEF6-EF03	Send untalk command on serial bus
EF04-EF18	Send unlisten command on serial bus
EF19-EFA2	Input a byte from serial bus
EFA3-EFED	NMI continue routine
EFEE-F035	Transmit byte

NAME:	FUNCTION:
F036-F173	NMI routine to collect data into bytes (RS-232)
F174-F1E1	Kernal messages
F1E2-F1F4	Print message to screen
F1F5-F20D	Get character from channel
F20E-F279	Input character from channel
F27A-F2C6	Output character to channel
F2C7-F308	Open channel for input
F309-F349	Open channel for output
F34A-F3EE	Close logical file
F3EF-F3F2	Close all logical files
F3F3-F409	Clear channels
F40A-F541	Open function
F542-F674	Load RAM function (from cassette or bus devices)
F675-F733	Save function
F734-F76F	Time function
F770-F77D	Test stop key
F77E-F7AE	Error handler
F7AF-F889	Find and read tape header
F88A-F98D	Cassette control routines
F98E-FABC	Tape read routines
FABD-FBE9	Byte handler for cassette read
FBEA-FD21	Tape write routines
FD22-FE90	System power up initialisation
FE91-FEA8	Memory check routines
FEA9-FF5B	NMI handler
FF5C-FF71	Baud rate tables
FF72-FF85	IRQ handler
FF85-FFFF	Kernal jump vector addresses

Waterloo Micro Software Description continued from page 25

loadable and executable progam file. Since it is disk-orientated, the Linker is capable of building programs which are larger than the RAM workspace available. The Linker supports the building of progams in segments or banks for operation in bank-switched RAM memory, as well as supporting the building of programs for operational in normal RAM memory.

d) The Waterloo microSUPERVISOR is an operating system designed for single-user microcomputer environments. It includes Monitor, Library and Serial Line Communications support as described below.

A Monitor progam is included which supports the loading of Linker-produced program files into bank-switched RAM memory or normal RAM memory. The Monitor also provides facilities which are useful for debugging machine-language programs. These include commands to display and alter RAM memory and 6809 microprocessor register, utilizing full-screen features for ease of use. In addition, a Monitor command permits disassembly of Motorola 6809 microprocessor instructions into assembly-language mnemonic form.

A Library of functions and procedures is included for general use by other programs included in the Package. The Library includes support functions for input/output operations to the keyboard, screen and peripheral devices. Other elements of the Library provide floating- point arithmetic, fundamental trigonometric functions, and several general-purpose utility functions.

A Serial Line Setup program is included which permits the selection of programmable characteristics, such as baud rate, of RS-232 serial lines. In addition, this program includes support for establishing communication with a host computer, through a serial line, for the purpose of accessing its files and peripheral devices.

A Cobol interpreter is currently nearing completion at Waterloo and will be available in the very near future. When details become available Commodore will immediately pass them on to the Users of the SP 9000.

SuperPET continued from page 29

REFERENCES:

1. *A.M. Addyman,* "A Draft Proposal for Pascal", *Pascal News 18 (May 1980)*
2. *M. Brown,* "ADA – the way ahead", Pacific Computer Weekly, (May 14-20, 1982) p. 8
3. W. Findlay and D. Watt, 'Pascal: An Introduction to Methodical Programming', *2 ed., Pitman (1981)*
4. *Gowans,* 'Do you need 16 bits?', *Microcomputer Printout (Dec. 1981) p. 64*
5. *C.A.R.Hoare,* 'The Emperor's Old Clothes', *Communications ACM 24 (Feb. 1981) p. 75*
6. 'COMPUTER', *IEEE (June 1981)*
7. *K. Jensen and N. Wirth,* 'Pascal User Manual and Report', *2 ed., Springer-Verlag (1978)*
8. 'Microcomputer Printout', *(Dec. 1981)*
9. *A. Osborne and C.S. Donahue,* 'PET/CBM Personal Computer Guide', *2 ed., OSBORNE/ McGraw Hill, Berkeley (1980)*
10. *H.D. Toong and A. Gupta,* 'An architectural Comparison of Contemporary 16-bit Microprocessors', *IEEE MICRO (May 1981) p. 26*
11. *P. Wegner,* 'Self-Assessment Procedure VIII', *Communications ACM 24 (Oct. 1981) p. 64*
12. *G. Wiliams,* 'A closer look at the IBM Personal Computer', *Byte (JAn. 1982) p. 37*

THE KERNAL ON THE VIC 20.

Essentially, the KERNAL is a standardized JUMP TABLE to the input, output, and memory management routines in the operating system. The locations of each routine in ROM might change as the system is upgraded, but the KERNAL entry points will not alter, only the jump table changed to match. If your machine language routines should see the system ROM routines through the KERNAL. The KERNAL is the operating system of the VIC computer. All input, output and memory management is controlled by the KERNAL.

A good question at this point is why use the jump table at all? The jump table is used so that if the KERNAL or BASIC is changed, your machine language programs will still work. In future operating systems the routines may be moved in memory ...but the jump table will still work correctly!

To use the KERNAL jump table, first you set up the parameters that the KERNAL routine needs to work. Then JSR to the proper place in the KERNAL jump table. After performing its function, the KERNAL transfers control back to your machine language program. Depending on which KERNAL routine you are using, certain registers may pass parameters back to your program. The particular registers for each KERNAL routine may be found in the individual descriptions of KERNAL subroutines.

That's all there is in using the KERNAL – these three steps :-

1. Set up
2. Call the routine
3. Error handling

USER CALLABLE KERNAL ROUTINES

NAME	ADDRESS HEX	ADDRESS DECIMAL	FUNCTION
ACPTR	$FFA	65445	Input byte from serial port
CHKIN	$FFC6	65478	Open channel for input
CHKOUT	$FFC9	65481	Open channel for output
CHRIN	$FFCF	65487	Input character from channel
CHROUT	$FFD2	65490	Output character to channel
CIOUT	$FFA8	65448	Output byte to serial port
CLALL	$FFE7	65511	Close all files
CLOSE	$FFC3	65475	Close logical file
CLRCHN	$FFCC	65484	Close input and output channel
GETIN	$FFE8	65512	Get character from keyboard queue
IOBASE	$FFF3	65523	Returns base address of I/O device
LISTEN	$FFB1	65457	Command Serial IEEE device to listen
LOAD	$FFD5	65493	Load RAM frorm device
MEMBOT	$FF9C	65436	Read/set bottom of memory
MEMTOP	$FF99	65433	Read/set top of memory
OPEN	$FFC0	65472	Open logical file
PLOT	$FFF0	65520	READ/SET X,Y cursor position
RDTIM	$FFDE	65502	Read real time clock
READST	$FFB7	65436	Read I/O status word
RESTOR	$FF87	65415	Restore old I/O vectors
SAVE	$FFD8	65496	Save RAM to device
SCNKEY	$FF9F	65439	Scan keyboard
SCREEN	$FFED	65517	Return X,Y organisation of screen
SECOND	$FF93	65427	Transmit secondary command
SETLFS	$FFBA	65466	Set logical, first, second address
SETMSG	$FF90	65424	Control KERNAL messages
SETNAM	$FFBD	65469	Set file name information
SETTIM	$FFDB	65499	Set real time clock
SETTMO	$FFA2	65442	Set timeout on serial bus
STOP	$FFE1	65505	Check stop key
TALK	$FFB4	65433	Command Serial IEEE device to talk
TKSA	$FF96	65430	Send secondary after talk
UDTIM	$FFEA	65514	Increment real time clock
UNTLK	$FFAB	65451	Command serial bus to UNTALK
VECTOR	$FF84	65412	Read/set vectored I/O

Further details on the Kernal routines can be found in either;
THE VIC REVEALED
by Nick Hampshire
VIC-20 PROGRAMMERS REFERENCE GUIDE
both available from your Commodore Dealer.

MAKE THE MOST OF YOUR COMMODORE

HIGH-RESOLUTION GRAPHICS

Now you can give your PET/CBM a High-Resolution Graphics capability with the MTU Graphics Hardware and Software Package. The Hardware is easily installed and the new graphics board provides five EXTRA ROM Sockets and 8k RAM MEMORY EXPANSION which can be used for program or data storage when graphics are not required. A powerful graphics Software Package is included and contains many extra BASIC commands for drawing lines defining shapes etc. The Graphics Hardware does not affect normal operation of the Commodore.

Available on all PET/CBM - BASIC 1, 2, 3, or 4.

RS232 TEST SET

The RS232 TESTSET is a small hand held device that connects inline with the interface cable, the terminal or the modem and monitors the line signals. The TESTSET passes all 25 lines through and so can be left connected without affecting communications. It is completely portable as no batteries are required since power us derived from the interface signals. Each indicator circuit is current limited to meet the requirements of the RS232 Interface Standard. Also the voltage range for activating the bright LED display corresponds with this standard and thereby reduces troubleshooting to a "GO-NOGO" problem instead of trying to measure active signals to determine voltage levels. One 2-pin and one 3-pin jumper are included.

PRINTOUT

Don't forget your PRINTOUT Magazine Subscription - for Commodore PET/CBM Lovers 12 issues p.a. from Jan. '82 incl. postage - $50.00 p.a.

VIC COMPUTING

For all VIC 20 owners, this sister to PRINTOUT is a must. 6 issues p.a. incl. postage from Jan. '82 - $25.00 p.a.

PROGRAMMERS TOOLKIT ROMS

These ROMs plug into spare sockets in your PET/CBMs and give the user additional commands such as TRACE, single STEP, FIND, RE-NUMBER, AUTO line numbering,DUMP variable contents, APPEND, and DELETE multiple lines. Also available are the DISK-O-PRO Tool-kits which gives all the extra DOS 2.0 commands to DOS 1.0 users as well as commands like PRINT USING, SCROLL and disk program MERGE -25 commands in all. The COMMAND-O provides DISK-O-PRO commands for BASIC 4.0 users.

The Programmers Toolkit for BASIC 1.0/2.0/3.0/4.0 users.
DISK-O-PRO Toolkit for BASIC 2.0/3.0 users.
COMMAND-O Toolkit for BASIC 4.0 users.

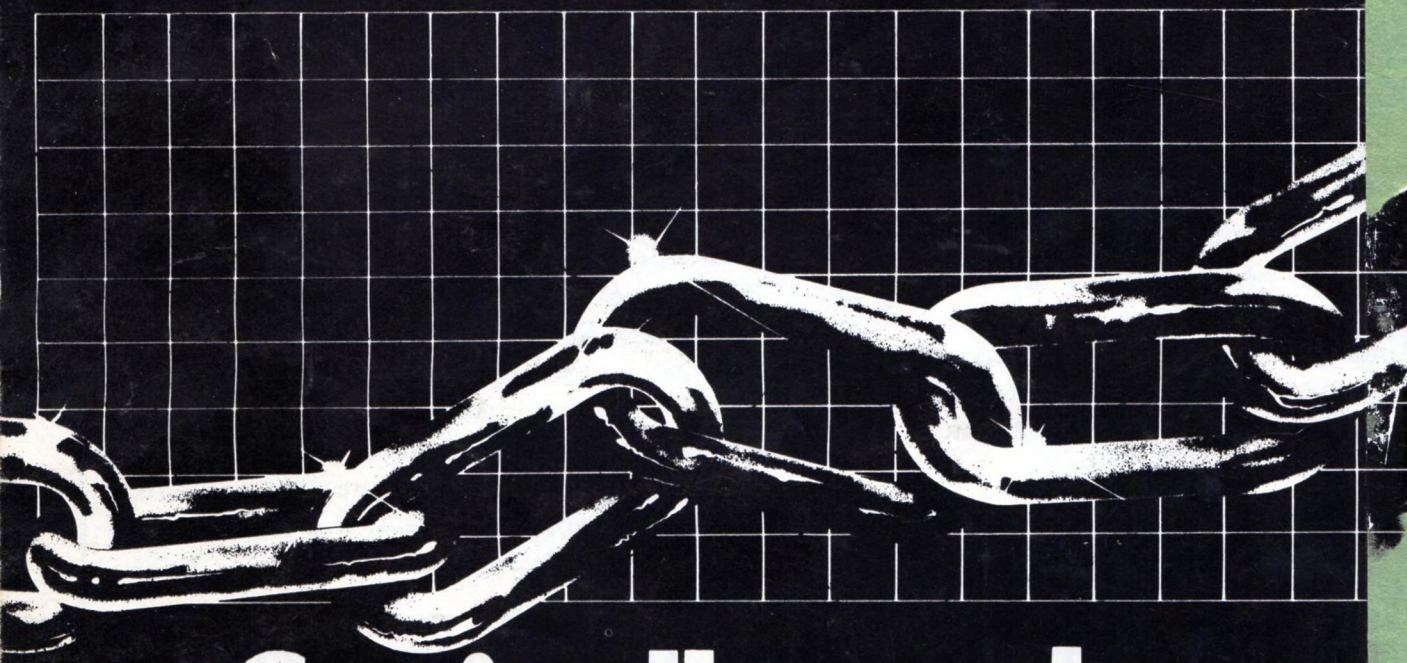
Tel: (03) 614 1433
614-1551
Telex: AA 30333.

MICROCOMP
MICROCOMPUTER SYSTEMS DESIGNERS

B.S. MICROCOMP PTY. LIMITED,
4th & 3rd Floors,
561 Bourke Street,
Melbourne, 3000.

The Link

THAT JOINS ALL OFFICE FUNCTIONS



Get it all together with the Silicon Office.

STORING AND RETRIEVING INFORMATION – CREATING, EDITING AND PRINTING OF TEXT – MATHEMATICAL CALCULATION – COMMUNICATING INFORMATION LONG DISTANCE.

Silicon Office is the first database management System for Commodore CBM Microcomputers whereby up to six files m open and accessed simultaneously during a run. It is alsc the first system which permits intercommunication with f machines and user. The Silicon Office turns the CBM 8032 into a secretarial work station capable of emulating any applic package the user cares to think of.

Now one program which is continuously and completely resident in the memory of the CBM is capable of performi functions required to run a small business or office. This can mean anything from Accounting and Stock Control to Processing, Statistical analysis, mailing lists and information filing – all at once, if necessary. Combine filing cabinets, le typewriter and calculator in your office into one efficient unit.

The Silicon Office package comprises of three integrated elements: a sophisticated word processor, a flexible dat management system and an option for inter computer communications – all in one memory resident program.

$8490

THIS BUSINESS PACKAGE IS NOW AVAILABLE AND WILL COMPRISE:

- 8023 DOT MATRIX/PSEUDO PRINTER
- COMMODORE CBM 8032 COMPUTER
- 8050 DISC DRIVE UNIT
- 64K ADD-ON MEMORY BOARD (TOTAL 96K RAM)
- THE SILICON OFFICE PROGRAM MASTER DISC
- TWO SECTIONAL A4 MANUALS

This package is available from:

Compute® CBM SYSTEMS

5 PRESIDENT AVE., CARINGBAH ☎ 525 5022